UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

Diplomarbeit

Evolutionäre Kreuzungsminimierung

Marko Tosic
31. Januar 2006

INTERNE BERICHTE
INTERNAL REPORTS

Diplomarbeit am
Fachbereich Informatik
der Universität Dortmund

Betreuer:
Prof. Dr. Petra Mutzel
Dipl.-Ing. Markus Chimani

Danksagung

Ich bedanke mich an dieser Stelle insbesondere bei:

Prof. Petra Mutzel dafür, dass sie zur richtigen Zeit am richtigen Ort war, um meine Begeisterung am automatischen Graphenzeichnen zu wecken und diese Arbeit überhaupt erst zu ermöglichen; Markus Chimani für die erstklassige Betreuung zu (fast) jeder Tages– und Nachtzeit; Karlheinz Schmitt für die unermüdliche Klärung aller meiner Fragen bzgl. evolutionärer Algorithmen; Thomas Bartz-Beielstein für sein geballtes Wissen über das Experimentieren und seine ausgezeichneten SPO-Skripte; Meiner Schwester Jana für das Korrekturlesen; Meiner Mutter für ihre seelische Unterstützung; Ulrich Hermes für die Erfüllung meiner administrativen Sonderwünsche.

All diejenigen, die ich an dieser Stelle vergessen habe, mögen mir verzeihen. Ihnen sei natürlich ebenfalls gedankt.

Kurzfassung

Diese Arbeit beschreibt einen Genetischen Algorithmus für das Planarisierungsproblem. welches von zentraler Bedeutung beim automatischen Zeichnen von Graphen ist. Das Planarisierungsproblem besteht darin, einen nicht-planaren Graphen durch Einfügen von Dummy-Knoten planar zu machen. Viele Algorithmen zum Zeichnen von Graphen gehen nach dem TSM-Modell vor. Dieses sieht in der ersten Phase die Planarisierung vor. Das dabei zu lösende Planarisierungsproblem ist NP-hart. Evolutionäre Algorithmen haben sich bereits für vielen NP-harte Probleme als geeigneter Lösungsansatz erwiesen. Auch die Planarisierung kann mittels evolutionärer Ansätze durchgeführt werden. Bei hinreichend langer Rechenzeit können dabei im Mittel weniger Dummy-Knoten als mit anderen Methoden erzielt werden. Für einige Anwendungen steht damit nun eine vielversprechende Alternative zu den bisher verwendeten Planarisierungsmethoden zur Verfügung.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Graphenzeichnen 1

1.2 Einordnung in den Kontext 2

2 Grundlagen **3**

2.1 Optimierungsprobleme 3

2.2 Evolutionäre Algorithmen 4

2.3 Graphen 15

2.4 Kreuzungsminimierung 22

2.5 Experimente 25

3 Evolutionäre Kreuzungsminimierung **35**

3.1 Evolutionäre Berechnung Maximaler Subgraphen 35

3.2 Evolutionäre Planarisierung 39

3.3 Kantengewichte 41

4 Experimente **45**

4.1 Die Rome-Library 45

4.2 Langzeitexperiment 47

4.3 Sequential Parameter Optimization 50

4.4 Vergleich der Varianten 55

4.5 Anwendung auf die Rome-Library 58

5 Fazit / Ausblick **63**

5.1 Fazit 63

5.2 Ausblick 63

Literaturverzeichnis **65**

Abbildungsverzeichnis

2.1 Links: Charles Darwin (1809-1882). Rechts: Gregor Mendel (1822-1884). [2]. 5

2.2 DNS-Moleküle. [2] 6

2.3 Übersicht: Evolutionäre Algorithmen (Quelle: [28]) 8

2.4 Oben: One-Point-Crossover. Unten: Two-Point-Crossover. 11

2.5 Crossover von Davis für Permutationen. 13

2.6 Links: PMX. Rechts: Order Crossover-2. 14

2.7 Links: Ungerichteter Graph. Rechts: Digraph. 16

2.8 Zwei verschiedene kombinatorische Einbettungen eines Graphen. Beim Einfügen der Kante $\{3, 7\}$ werden unterschiedlich viele neue Kreuzungen erzeugt [16]. 18

2.9 Ein ebener Graph mit dazugehörigem dualen Graphen. 19

2.10 Topology-Shape-Metric Verfahren. 21

2.11 Vorgehen beim Einfügen einer Kante bei fixer kombinatorischer Reihenfolge. 24

2.12 Ein Histogramm 27

2.13 Ein Boxplot. Der Median ist als breite Linie innerhalb des Kastens dargestellt. 28

3.1 Beispiel für zwei verschiedene Einfügereihenfolgen. 1: Der Ursprungsgraph. 2: Die Kantennummerierung. 3: Maximaler planarer Subgraph zu Permutation $(7, 12, 10, 6, 3, 1, 4, 9, 2, 5, 11, 8)$. 4: Maximaler planarer Subgraph zu Permutation $(12, 10, 7, 8, 4, 1, 3, 5, 2, 9, 11, 6)$. 37

3.2 Mögliche unterschiedliche Ergebnisse der Varianten FIX und VAR. 1: Ungünstiges Ergebnis für FIX. 2: Besseres Ergebnis für VAR. 41

3.3 Die Kantengewichte, wie sie für $\epsilon = 0.1$ (oben) bzw. $\epsilon = 0.2$ (unten) generiert werden. 42

4.1 Verteilung der Knotenanzahlen in der Rome-Library. 46

4.2 Verteilung der mittleren Kantenanzahlen in der Rome-Library. 46
4.3 Verlauf der Fitnesswerte ausgewählter Graphen bei der Optimierung. . 51
4.4 Differenzen der Kreuzungsanzahlen 54
4.5 Übersicht der Ergebnisse der vier Varianten. Von oben nach unten: VAR, FIX, WVAR, WFIX 57
4.6 Verteilung der Ergebnisse bei den Varianten VAR und FIX. 58
4.7 Verteilung der Knotenanzahlen unter den Graphen der Testmenge . . 59
4.8 Verteilung der durchschnittlichen Kantenanzahl unter den Graphen der Testmenge 59
4.9 Darstellung der mittleren Anzahl von Kreuzungen aufgeteilt nach der Knotenanzahl der zugrunde liegenden Graphen. 61
4.10 Anzahl der Graphen nach Anzahl der Kreuzungen. 62

Kapitel 1

Einleitung

Dieses Kapitel gibt eine kurze Übersicht über das automatische Graphenzeichnen und ordnet die Arbeit in den Kontext ein.

1.1 Graphenzeichnen

Ein Graph ist eine Datenstruktur, die Relationen zwischen Objekten speichert. Graphen spielen eine wichtige Rolle in der Informatik. Kein Informatiker kommt an ihnen vorbei. Dies liegt nicht zuletzt daran, dass sie so universell einsetzbar sind. Viele Probleme lassen sich auf natürliche Weise als Graphenprobleme formulieren. Dies hat zu einer weiten Verbreitung von Graphen in den verschiedensten Disziplinen geführt. Hier sind nur einige Beispiele:

1. Logistik. Graphen können eingesetzt werden, um Orte und Verbindungen dazwischen darzustellen.
2. Softwaredesign. Graphen können Diagramme darstellen, wie beispielsweise UML-Diagramme und Flussdiagramme.
3. Platinenentwurf. Graphen können Bauteile und dazwischenliegende Leiterbahnen repräsentieren.
4. Datenbanken. Graphen können Verbindungen zwischen Datensätzen repräsentieren.

Diese Vielseitigkeit macht Graphen so interessant, insbesondere auch für die Forschung. Dies führte zu einer fundierten Theorie über Graphen und nicht zuletzt zu zahlreichen effizienten Graphen-Algorithmen.

Ein Grund für die Beliebtheit von Graphen ist, dass man sie sich gut vorstellen kann; man kann sich von ihnen ein Bild machen. Aber gerade dies, das Erstellen eines Bildes eines Graphen, hat sich für Computer als schwieriger Aufgabe erwiesen. Das Spezialgebiet des Graphenzeichnens nimmt sich dieser Aufgabe an.

Die meisten Algorithmen, die für das Zeichnen von Graphen entwickelt wurden, beruhen auf der Planrisierung: Ein nicht-planarer Graph wird zunächst durch das Einfügen von Dummy-Knoten planar gemacht. Der planarisierte Graph wird anschließend gezeichnet.

In dieser Arbeit wird ein evolutionärer Ansatz verfolgt, um die Planarisierung durchzuführen.

1.2 Einordnung in den Kontext

Obwohl es vereinzelt Ansätze gibt, Graphen mit Hilfe evolutionärer Verfahren zu zeichnen, ist kein Verfahren bekannt, dass speziell die Planarisierung evolutionär angeht.

Branke, Bucher und Schmeck verwenden ein Genetischen Algorithmus in Verbindung mit einem Spring-Embedder-Algorithmus um Straight-Line Zeichnungen von Graphen zu erstellen [9].

Michalewicz beschreibt ebenfalls einen Genetischen Algorithmus zum Zeichnen von Graphen [26].

Rodriguez und Suarez verwenden Evolutionsstrategien um Graphen zu zeichnen. Die evolutionären Operatoren arbeiten dabei direkt auf den Koordinaten der Knoten [32].

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen eingeführt, die für das Verständnis der Arbeit von Bedeutung sind. In Abschnitt 2.1 wird die Klasse der Optimierungsprobleme vorgestellt, zu der auch das in dieser Arbeit untersuchte Kreuzungsminimierungsproblem gehört. Der Abschnitt 2.2 widmet sich den Grundlagen evolutionärer Algorithmen. Nach einer kurzen Darstellung der biologischen Evolution werden die wichtigsten Ansätze beschrieben, die versuchen, durch simulierte Evolution Optimierungsprobleme zu lösen. Der besondere Augenmerk liegt dabei auf den genetischen Algorithmen, da diese später in der Arbeit angewendet werden. Der darauf folgende Abschnitt 2.3 beschäftigt sich mit Graphen. Es werden zunächst graphentheoretische Grundlagen eingeführt, mit deren Hilfe anschließend das Kreuzungsminimierungsproblem definiert und der Begriff der Planarisierung erläutert wird. Es folgt eine Darstellung bekannter Lösungsansätze für dieses Problem. Zum Abschluss des Grundlagenkapitels wird in Abschnitt 2.5 erklärt, wie ein experimentelles Design nach der Methode „Design of Experiments“ aufgestellt und ausgewertet wird.

2.1 Optimierungsprobleme

Das Kreuzungsminimierungsproblem, dem sich diese Arbeit widmet, gehört zu der Klasse der *Optimierungsprobleme*. Wir werden deshalb diese Problemklasse zunächst etwas genauer spezifizieren. Dabei folgen wir der Darstellung von Gerdes et al. [19].

Definition 2.1. *Ein Optimierungsproblem ist definiert durch seinen* Suchraum $\mathcal{S}$ *und seine Zielfunktion* f*. Die Menge* $\mathcal{S}$ aller *gültigen Lösungen eines Problems heißt ihr* Suchraum*. Eine Zielfunktion ist eine Funktion* $f : \mathcal{S} \to \mathbb{R}$*, die jeder Lösung des Suchraums ihre Bewertung zuordnet. Das Paar* $(\mathcal{S}, f)$ *bezeichnen wir als Optimierungsproblem.*

Bei den Optimierungsproblemen unterscheidet man zwischen Maximierungsproblemen und Minimierungsproblemen.

Definition 2.2. *Ein Maximierungsproblem ist ein Optimierungsproblem, bei dem ein Wert* $o \in \mathcal{S}$ *so gesucht wird, dass für alle* $x \in \mathcal{S}$ *gilt:* $f(o) \geq f(x)$*. Analog dazu wird*

bei einem Minimierungsproblem ein Wert $u \in \mathcal{S}$ so gesucht, dass für alle $x \in \mathcal{S}$ gilt: $f(u) \leq f(x)$. Wir bezeichnen den Wert o bzw. u als globales Optimum oder spezifischer als globales Maximum bzw. globales Minimum. Gibt es mehrere solcher Werte, so spricht man von globalen Optima, genauer von globalen Maxima bzw. globalen Minima.

Für viele Optimierungsprobleme gibt es eine *Entscheidungsvariante*. Statt nach einer optimalen Lösung zu suchen soll entschieden werden, ob eine vorgegebene Lösung erreicht oder unterboten werden kann. Formal wird also die Frage gestellt, ob es zu einer vorgegebenen Bewertung $y \in \mathbb{R}$ eine Lösung $s \in \mathcal{S}$ gibt, so dass $f(s) \leq y$ gilt. Oft ist bereits diese Entscheidungsvariante „schwer“ zu beantworten, sie ist NP-vollständig. Wir wollen an dieser Stelle den Begriff der NP-Vollständigkeit nicht genauer beschreiben und verweisen stattdessen auf die Literatur zu den Grundlagen theoretischer Informatik [35].

Die Schwierigkeit der Optimierungsprobleme macht ein vollständiges Durchsuchen des Lösungsraumes nach einer optimalen Lösung bei einem entsprechend großen Suchraum unmöglich. Im Laufe der Zeit wurden zahlreiche Techniken entwickelt, die es dennoch erlauben gute, wenn auch nicht optimale Lösungen, zu berechnen. Beispielsweise findet sich eine Zusammenstellung solcher Verfahren für das populäre „Traveling Salesman Problem“ in [23].

Oft kann es von Vorteil sein, anstelle des eigentlichen Suchraums $\mathcal{S}$, der alle gültigen Lösungen enthält, einen größeren, einfacher strukturierten Suchraum $\hat{\mathcal{S}}$ zu erlauben. Allerdings muss dann beachtet werden, dass dieser auch ungültige Lösungen enthält. Um diese zu vermeiden, bieten sich zwei Techniken an:

1. Falls es das verwendete Optimierungsverfahren erlaubt, Nebenbedingungen zu verwenden, so können diese so formuliert werden, dass sie nur von korrekten Lösungen erfüllen werden.
2. Die Zielfunktion wird um einen Term erweitert. Dieser „bestraft“ unkorrekte Lösungen, indem er deren Funktionswerte verschlechtert. Man spricht in diesem Fall von einer *Straffunktion* und von *Strafkosten*, die zum Zielfunktionswert beispielsweise hinzuaddiert werden.

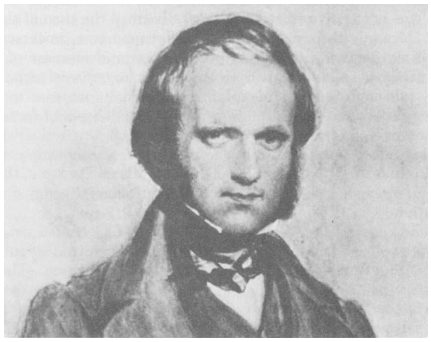
Es sei darauf hingewiesen, dass bei einigen Problemen die Wahl der Zielfunktion nicht eindeutig ist. Allerdings wird uns dies im Rahmen dieser Arbeit nicht beschäftigen.

Im Weiteren wird hauptsächlich eine bestimmte Art von Optimierungsproblemen betrachtet, die *kombinatorischen Optimierungsprobleme*.

Definition 2.3. *Ein kombinatorisches Optimierungsproblem ist ein Optimierungsproblem $(\mathcal{S}, f)$, bei dem $S \in \mathbb{N}^n$ ist. Die Lösungen bestehen also aus Vektoren diskreter Werte.*

2.2 Evolutionäre Algorithmen

Evolutionäre Algorithmen stellen eine Simulation von natürlichen evolutionären Vorgängen dar. Aus diesem Grund werden zunächst einige Grundlagen aus der Biologie



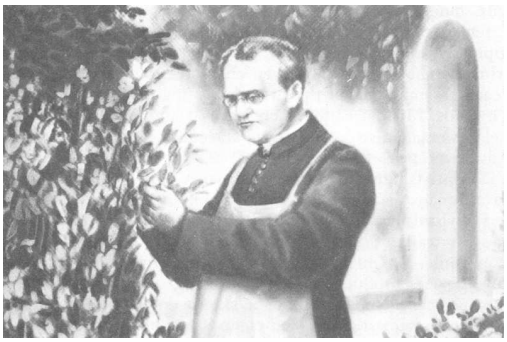


Abbildung 2.1: Links: Charles Darwin (1809-1882). Rechts: Gregor Mendel (1822-1884). [2].

vorgestellt. Diese sollen es dem Leser erleichtern, die anschließende kurze Übersicht der evolutionären Algorithmen und die darauf folgende genauere Beschreibung von genetischen Algorithmen zu verstehen.

2.2.1 Biologische Evolution

Möchte man verstehen, wie es zu der Idee für evolutionäre Algorithmen kam, hilft es, sich einige Grundkenntnisse der biologischen Evolution anzueignen. Aus diesem Grund werden in diesem Abschnitt die wichtigsten Grundlagen kurz erläutert. Die Darstellung verläuft dabei analog zu der von Nissen [28]. Für Fachbegriffe, die aus der Genetik stammen, sei auf das Fachwörterbuch von King und Stansfield [21] verwiesen.

Die Grundlage des Evolutionsmodells, wie es heute verwendet wird, bildet die Arbeit Darwins. Wallace kam in seinem Essay „Über die Tendenz der Varietäten unbegrenzt von dem ursprünglichen Typus abzuweichen" zeitgleich auf dieselben Erkenntnisse, dennoch wird im Allgemeinen von der darwinistischen Theorie gesprochen [1]. Darwin war bei seinen Untersuchungen aufgefallen, dass obwohl in der Natur zu viele Lebewesen produziert werden, die Populationsgröße nicht ansteigt. Er folgerte daraus, dass die meisten der produzierten Individuen noch vor der Fortpflanzung sterben, da sonst ein exponentielles Wachstum der Population erkennbar sein müsste. Im Laufe der Zeit wurden Darwins Ideen verfeinert und erweitert, insbesondere um Erkenntnisse aus der Genetik, als deren Begründer Mendel gilt.

Hier wird der Neo-Darwinismus beschrieben, welcher den Ideen evolutionärer Algorithmen wie sie im Folgenden beschrieben werden am nächsten kommt.

Wir beginnen die Einführung mit einigen Grundbegriffen, die im Zusammenhang mit Evolution von Bedeutung sind. Anschließend werden die wichtigsten Mechanismen der natürlichen Anpassung erläutert.

Die gesamten Erbinformationen eines Lebewesens sind in den so genannten Chromosomen gespeichert. Diese sind Stränge aus Molekülen der Desoxyribonukleinsäure (*DNS*).

Die Positionen (*locus*) auf diesem Chromosom, die eine einzelne Informationseinheit enthalten, werden als Gen bezeichnet. Eine mögliche Ausprägung eines Gens ist ein

Allelomorph oder kurz Allel. Beispielsweise könnte es ein Gen für das äußere Merkmal „Augenfarbe“ geben mit den Allelen „blau“, „grau“, usw.

Grundsätzlich unterschieden wird zwischen *Phenotyp* und *Genotyp* eines Lebewesens. Der Genotyp benennt die Gesamtheit aller Gene einer Lebensform, also bildlich gesprochen dessen „Bauplan“. Der Phenotyp hingegen beschreibt das äußere Erscheinungsbild eines bestimmten Lebewesens.

Höhere Lebensformen haben häufig zwei komplette Chromosomensätze, sie werden als *diploid bezeichnet* (im Gegensatz zu *haploid*). In diesem Fall sind für jedes Gen zwei Allele vorhanden. Haben diese dieselbe Ausprägung, so spricht man vom *homozygoten* Zustand, haben sie unterschiedliche Ausprägungen, so spricht man vom *heterozygoten* Zustand. Auch wenn zwei Allele für ein Gen vorhanden sind, so bestimmt doch nur eines von diesen, das *dominante*, das Erscheinungsbild. Das andere heißt das *rezessive* Allel. Durch die Diploidie wird der Rückschluss vom Phenotyp auf den Genotyp erschwert.

Hinzu kommt, dass oftmals mehrere Gene an der Ausbildung eines äußeren Merkmals beteiligt sind. In diesem Fall spricht man von *Polygenie*. Hat ein äußeres Merkmal genau ein Gen, so nennt man dies *Pleiotropie*.

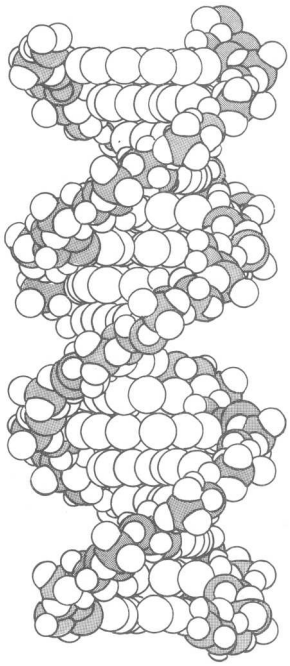


Abbildung 2.2: DNS-Moleküle. [2]

Mutation

Unter einer *Mutation* versteht man eine spontane Veränderung im Erbmaterial. Durch diese Veränderung kann es manchmal zu phänotypisch wirksamen Varianten des Grundtyps einer Art kommen. Nissen bezeichnet Mutationen deshalb auch als “Materiallieferanten der Evolution” ([28, S. 5]). Man unterscheidet die folgenden Mutationsarten:

- Bei einer *Genmutation* verändert sich die Ausprägung eines einzelnen Gens. Dies kommt sehr selten vor.
- Bei der *Chromosomenmutation* wird ein Teilstück des Chromosoms abgetrennt oder invertiert.
- Bei der *Genommutation* wird die Anzahl einzelner Chromosomen oder die Anzahl ganzer Chromosomensätze verändert.

Selektion

Unter der *Selektion* versteht man die Reduzierung der Individuenanzahl einer Population. Dies ist ein stochastischer Prozess, bei dem die bestangepassten Individuen bevorzugt werden. Aber auch weniger gut angepasste Individuen haben eine wenngleich

auch geringere Chance, zu überleben. Durch die Selektion ist es also wahrscheinlicher, dass besser angepasste Individuen ihr Erbmaterial auch an die nachfolgende Generation weitergeben. Darwin zufolge erfolgt die natürliche Selektion vor der Reproduktion. Dabei ist zu beachten, dass die Lebewesen in ein komplexes Ökosystem eingebunden sind, so dass noch andere Faktoren auf das Überleben der Individuen einwirken.

Mutation und Selektion zusammengenommen bilden nun einen Mechanismus zur Anpassung an dynamische Lebensbedingungen, die eine sich ständig ändernde Umwelt mit sich bringt. Durch die Mutation werden neue Variationen einer Art erzeugt, die bei der Selektion ihre Angepasstheit unter Beweis stellen müssen. Bewährt sie sich dabei, geht sie in den Genpool ein, andernfalls wird sie verworfen. So passt sich die Art Schritt für Schritt an seine Umwelt an.

Rekombination

Mutation und Selektion sind nicht die einzigen Methoden, nach denen sich eine Art an seine Umwelt anpassen kann. Bei der geschlechtlichen Fortpflanzung kommt es zu einer Neukombination des Erbmaterials der elterlichen Gene. Man nennt diesen Vorgang *Rekombination.* Durch die Rekombination bietet sich die Möglichkeit, die vorteilhaften Allele zweier Individuen in den Nachkommen zu vereinen. Dadurch gewinnt die Nachkommenschaft, ähnlich wie bei der Mutation, einen potenziellen Vorteil bei der Selektion.

2.2.2 Simulierte Evolution

Die optimierenden Eigenschaften der biologischen Evolution, wie sie im letzten Abschnitt beschrieben wurden, können nutzbar gemacht werden, um schwierige Optimierungsaufgaben zu lösen.

Die Idee biologische Evolution für technische Aufgaben anzuwenden ist vergleichsweise alt. So beschreibt beispielsweise Box schon 1957 [7] eine Technik, bei der eine Fabrik als eine sich entwickelnde Spezies betrachtet wird, die durch eine Reihe von zufälligen Mutationen und Selektionen die Qualität der produzierten Produkte schrittweise verbessert.

Friedberg et al. beschreiben [11, 12] wie ein Maschinenprogramm für einen simulierten Rechner mit Hilfe eines weiteren Programms entwickelt werden kann. Damit gelang es zu einem gewissen Grad, einfache Programme zu entwickeln.

Die heute unter dem Begriff „Evolutionäre Algorithmen“ zusammengefassten Verfahren sind Evolutionsstrategien, evolutionäre Programmierung, genetische Algorithmen und genetische Programmierung (Abb. 2.3).

Wir zeigen an dieser Stelle kurz die Ideen von Evolutionsstrategien, Evolutionärer Programmierung und Genetischer Programmierung auf. Genetische Algorithmen werden im nächsten Abschnitt ausführlicher behandelt, da sie im weiteren Verlauf der Arbeit angewendet werden.

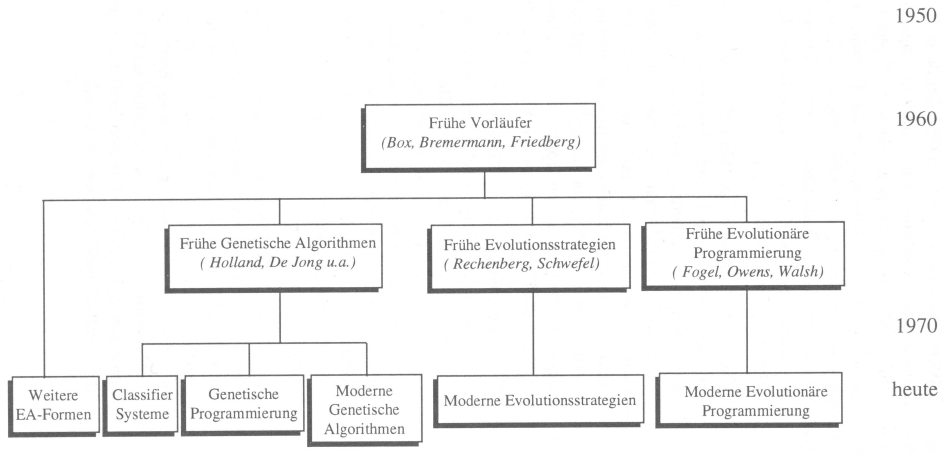


Abbildung 2.3: Übersicht: Evolutionäre Algorithmen (Quelle: [28])

Evolutionsstrategien

Evolutionsstrategien (ES) wurden mitte der 60er Jahre von Rechenberg und Schwefel entworfen, um Optimierungsaufgaben aus dem technisch-pysikalischem Bereich zu lösen. In der Grundform dienen die ES zur Optimierung von reellwertigen Zielfunktionen, also Funktionen $f : \mathbb{R}^n \to \mathbb{R}$.

Zu Beginn wird eine Startpopulation erzeugt, also eine Menge $P(0)$ von Lösungen $\vec{a_i} \in \mathbb{R}^n$ der Zielfunktion, die auch als *Individuen* bezeichnet werden. Diese kann entweder zufällig gleichverteilt über den gesamten Lösungsraum generiert werden oder, falls vorab genügend Informationen verfügbar sind, gezielt mit geeigneten Lösungen initialisiert werden. Die Größe dieser Menge, die sogenannte *Populationsgröße* wird bei ES mit μ abgekürzt.

Die Population wird nun über eine Anzahl Generationen mit Hilfe von evolutionären Prozessen verändert. Aus der Generation $P(t)$ wird die Generation $P(t+1)$ mit Hilfe von der natürlichen Evolution nachempfundenen Operatoren zur Mutation und Rekombination erzeugt. In der Grundvariante der ES kommt dabei jedoch nur die Mutation zum Einsatz. Es wird in jeder Generation eine Menge von Nachkommen aus der Population erzeugt. Dies geschieht, indem zufällig ein Individuum $\vec{a_i}$ aus der Population als Elter ausgewählt und kopiert wird. Der Nachfahre sei $\vec{b_j}$. Dieser wird anschließend mutiert, indem ein Vektor $\vec{c_j} = (c_{j0}, \ldots, c_{j\lambda-1}) \in \mathbb{R}^n$ hinzuaddiert wird, wobei die c_{jk} zufällig gewählt werden. Dabei ist zu beachten, dass die Nachfahren „in der Nähe“ der Eltern liegen. Wie weit die einzelnen Werte auseinander liegen dürfen, wird durch die *Mutationsschrittweite* begrenzt. Anschließend werden aus den Eltern und den Nachfahren die Individuen der nächsten Generation ausgewählt. Hierfür gibt es zwei verschiedene Strategien: die *Plus-Strategie* und die *Komma-Strategie*.

- Bei der Plus-Strategie werden die μ bestangepassten Individuen aus der Vereinigung der Mengen der Eltern und der Nachkommen ausgewählt. Als bestan-

gepasst gelten dabei diejenigen Individuen, die bei einem Minimierungsproblem gemäß der Zielfunktion den kleinsten Funktionswert besitzen bzw. bei einem Maximierungsproblem den größten Funktionswert. Man spricht in diesem Fall von einem $(\mu + \lambda) - ES$.

- Bei der Komma-Strategie wird die Nachfolgegeneration ausschließlich aus den besten Individuen der Nachkommen gebildet, die Eltern werden nicht berücksichtigt. In diesem Fall spricht man von einem $(\mu, \lambda) - ES$.

Die Populationsgröße μ, die Anzahl der generierten Nachfahren λ und die Mutationsschrittweite sind dabei Strategieparameter, die den Verlauf der Optimierung stark beeinflussen und daher mit Bedacht ausgewählt werden müssen bzw. durch Experimente geeignet eingestellt werden müssen.

Im Laufe der Zeit wurden diese beiden Grundarten der ES durch zahlreiche Variationen ergänzt. So kann beispielsweise die Mutationsschrittweise dynamisch angepasst werden. Auch die Ergänzung um einen Rekombinationsoperator ist geläufig, so dass mehre Eltern einen Nachkommen erzeugen, der anschließend noch mutiert.

Genetische Programmierung

Die Genetische Programmierung (GP) ist ein Verfahren der Programm-Induktion, also eine Methode, die mit Hilfe von Trainingsdaten ein Programm automatisch erzeugt. Ein früher Ansatz in diese Richtung war der Lernautomat von Friedberg [12]. Der Durchbruch kam jedoch erst später, insbesondere durch die Arbeit von Koza, die 1992 veröffentlicht wurden und der GP ihren Namen gab [22].

Die Individuen bei GP sind Computerprogramme. Diese werden aus geeigneten Funktionen, Variablen und Konstanten zusammengesetzt. Koza verwendet hierfür eine für GP angepasste Variante der Programmiersprache LISP. Diese hat den Vorteil, dass sie eine interpretierte Sprache ist, die nicht zwischen Daten und Programmen unterscheidet.

Um gute Ergebnisse zu erhalten, muss die verwendete Sprache in geeigneter Weise festgelegt werden. Dies geschieht über die Auswahl einer geeigneten Menge von Funktionen sowie einer geeigneten Menge von Terminalsymbolen. Die Funktionen sollten so gewählt werden, dass sie das rasche Auffinden von guten Lösungen erlauben. Dabei ist zu beachten, dass die Größe der beiden Mengen die Größe des Suchraums definiert. Daher ist es besser, sich auf einige wenige ausdrucksstarke Funktionen zu beschränken als möglichst viele in die Menge aufzunehmen. Die Terminalsymbole sind die Variablen und Konstanten, die das Lösungsprogramm verwenden kann. Auch hier ist eine angemessene Auswahl von Elementen wichtig.

Weiterhin muss ein Maß für die Fitness der Individuen, also der erzeugten Programme, festgelegt werden. Dieses sollte den Anforderungen der Anwendung gerecht werden. Oftmals kann beispielsweise ein Fehlerterm verwendet werden, der die Differenz der Ausgabe des Programms von der korrekten Ausgabe über eine Menge von Testeingaben aufsummiert. Weitere Kriterien, die für die Gestaltung des Fitnessmaßes herangezogen werden können, sind zum Beispiel die Programmqualität und die Programmkomplexität, die mit geeigneten Verfahren festgestellt werden kann.

Evolutionäre Programmierung

Die Evolutionäre Programmierung (EP) wurde von L.J. Fogel et al. in den 60er-Jahren eingeführt. Das Ziel war zu diesem Zeitpunkt jedoch nicht das Lösen von Optimierungsproblemen, sondern die Entwicklung von endlichen Automaten, die über eine künstliche Intelligenz verfügen. Auf die Forschung seines Vaters aufbauend entwickelte D.B. Fogel später die EP zu der heute bekannten Form weiter. Damit steht nun auch die evolutionäre Programmierung zur Lösung von Optimierungsproblemen bereit.

EP ist ein Sonderfall unter den Evolutionären Algorithmen. Es wird von der sonst üblichen genotypischen Analogie zur biologischen Evolution abgewichen und stattdessen eine phänotypische Sichtweise angenommen. Dies bedeutet, dass die Individuen nun nicht mehr den genetischen Code repräsentieren, sondern die daraus hervorgehenden äußeren Merkmale.

Die Grundannahme bei EP ist also, dass die evolutionäre Anpassung auf der Ebene des Verhaltens stattfindet. Ein Individuum des EP repräsentiert also nicht wie bei den anderen EAs den genetischen Code eines Individuums einer Population, sondern abstrakter eine ganze Population von Individuen bzw. eine Art von Lebenwesen und Ihre Verhaltensweisen. Als Evolution wird hier die Anpassung der Art an ihre Umwelt angesehen. Daraus folgt, dass Operatoren auf genetischer Ebene wie beispielsweise ein Crossover oder eine Inversion hier keinen Sinn ergeben.

Die Hauptanwendung für EP liegt in der Optimierung von Optimierungsproblemen mit kontinuierlichen Variablen. Die Menge aller möglichen Lösungen ist in diesem Fall also $S = \mathbb{R}^n$ und die Zielfunktion des Optimierungsproblems daher $f : \mathbb{R}^n \to R$.

2.2.3 Genetische Algorithmen

Die Genetischen Algorithmen (GA) wurden seit Beginn der 60er Jahre von J.D. Holland entwickelt und 1975 veröffentlicht [17]. Populär wurden GAs allerdings erst später durch die Arbeit von Goldberg [13]. Wir beschreiben zunächst den Standard-GA und die dafür typischen Operatoren. Davon ausgehend werden anschließend einige spezielle Operatoren vorgestellt, die später in der Arbeit eingesetzt werden.

Standard-GA

Eine vereinfachte Form der biologischen Evolution diente als Inspiration für den Standard-GA. Es werden Selektion und Rekombination simuliert. Dazu werden die Lösungen des betrachteten Optimierungsproblems als genetischer Code aufgefasst, der in Form von Bitstrings gespeichert wird. Diese stellen somit das Pendant zu den Chromosomen dar. Es ist wichtig, sich Gedanken über die Repräsentation der Lösung als Bitstring zu machen. Ganze Zahlen können beispielsweise neben der üblichen Binärdarstellung für ganze Zahlen auch im Graycode gespeichert werden. Reelle Zahlen können als Festkomma- oder als Fließkommazahlen mit unterschiedlicher Genauigkeit gespeichert werden. Die Wahl der Repräsentation kann große Auswirkungen auf den Verlauf der Optimierung haben.

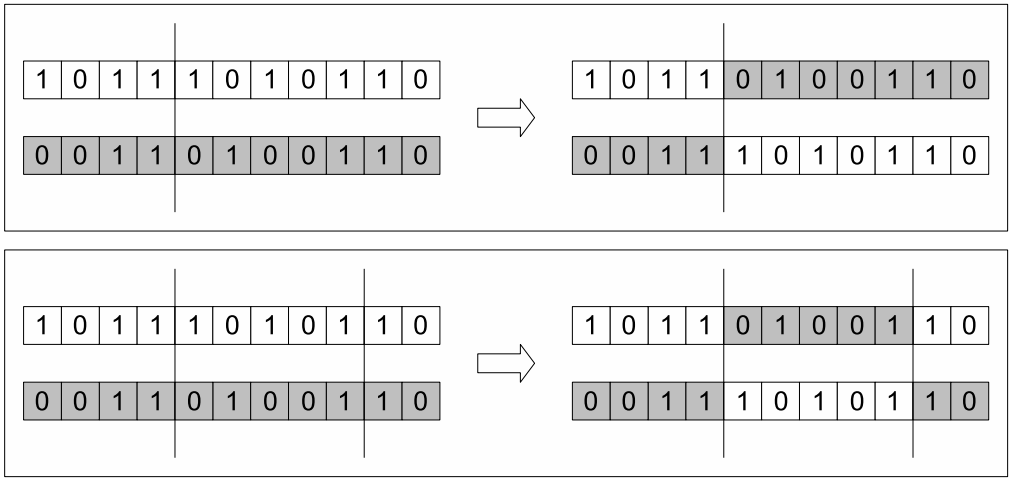


Abbildung 2.4: Oben: One-Point-Crossover. Unten: Two-Point-Crossover.

Neben der Darstellung als Bitstrings ist für GAs der Crossover-Operator typisch. Die einfachste Form ist der One-Point-Crossover Operator 2.4. Es wird zufällig ein Kreuzungspunkt ausgewählt. Es werden nun zwei neue Bitstrings zusammengesetzt, indem von einem Bitstring der Teil bis zum Kreuzungspunkt kopiert wird und vom anderen Bitstring der Rest. Dies ist in Abbildung 2.4 oben dargestellt. Im unteren Teil der Abbildung ist der *Two-Point-Crossover* Operator dargestellt, bei dem ein weiterer Kreuzungspunkt hinzukommt. Dadurch wird ein zufällig gewählter Teil aus der Mitte der beiden Bitstrings ausgetauscht.

Erweiterungen des Standard-GAs

Im Laufe der Zeit wurden zahlreiche Varianten des Standard-GAs untersucht mit dem Ziel, die Leistungsfähigkeit der Optimierung zu steigern. Da GAs nicht die einzigen evolutionären Algorithmen waren, die untersucht wurden, lag es nahe, Operatoren, die eigentlich anderen EAs zugeordnet werden, auch in GAs einzusetzen. So ist es mittlerweile üblich, auch einen Mutationsoperator zu implementieren. Dieser kann beispielsweise implementiert werden, indem einfach ein zufälliges Bit des Bitstrings *flippt*, also seinen Wert von 0 auf 1 ändert, bzw. von 1 auf 0. Andere Erweiterungen sehen vor, nicht nur Bitstrings, sondern spezielle Vektoren von computerimplementierten Typen zu verwenden. Die Idee dahinter ist, die zu einer Variablen des Problems gehörenden Bits enger aneinander zu binden.

Ablauf eines GA

Der Ablauf des genetischen Algorithmus lässt sich nun wie folgt darstellen:

1. Initialisierung: Es wird eine Population von Individuen erzeugt. Die Populationsgröße ist der dabei ein Laufzeitparameter, dessen Wahl großen Einfluss auf

den Verlauf der Optimierung haben kann. Die Individuen werden mit Hilfe der Fitnessfunktion bewertet.

2. Solange das Abbruchkriterium nicht erfüllt ist, geht man wie folgt vor:

   (a) **Selektion**: Wähle eine Menge von Individuen aus, welche für die Erzeugung der Nachkommenschaft herangezogen werden. Diese gehen in den sogenannten *Mating-Pool* ein. Die Anzahl der so ausgewählten Individuen ist der nächste Laufzeitparameter. Man spricht von der *Poolgröße*.

   (b) **Rekombination**: Es werden nun je zwei Individuen, die *Eltern*, aus dem Mating-Pool betrachtet. Mit einer gewissen Wahrscheinlichkeit, der *Crossover-Wahrscheinlichkeit*, werden diese nun mittels des gewählten Crossover-Operators miteinander verknüpft. Die Crossover-Wahrscheinlichkeit ist ein weiterer Laufzeitparamter. So entstehen die Nachfahren, die nächste Generation. Die Wahrscheinlichkeit, dass eine Verknüpfung mit dem Crossover-Operator stattfindet, ist dabei groß (Richtwert: $0,8$).

   (c) **Mutation**: Falls Mutation eingesetzt wird, so werden die Nachfahren mit einer gewissen Wahrscheinlichkeit, der *Mutationswahrscheinlichkeit* durch den Mutationsoperator verändert. Die Mutationswahrscheinlichkeit ist ebenfalls ein Laufzeitparameter. Im Allgemeinen ist diese Wahrscheinlichkeit sehr gering (Richtwert: $0,001 - 0,01$).

   (d) Die neue Generation wird mit Hilfe der Fitnessfunktion bewertet.

   (e) **Selektion**: Es wird nun festgelegt, welche Individuen aus den beiden betrachteten Generationen überleben. Die Wahrscheinlichkeit des Überlebens eines Einzelindividuums ist dabei i.A. proportional zu seinem Fitnesswert.

3. Das Individuum mit dem besten Fitnesswert repräsentiert die beste gefundene Lösung. Diese wird ausgegeben.

Permutationen als Individuen

Neben dem oben beschriebenen Standard-GA haben sich im Laufe der Zeit eine Reihe von Varianten für die verschiedenen Optimierungsaufgaben entwickelt. Eine davon ist hier von besonderem Interesse, nämlich diejenige, die Permutationen als Individuen verwendet. Diese eignen sich besonders gut, um kombinatorische Optimierungsaufgaben zu bearbeiten, wie es hier geschehen soll. Whitley weißt jedoch darauf hin, dass die Verwendung von Permutation einige Schwierigkeiten in sich birgt, die besonderer Aufmerksamkeit bedürfen. [36] .

Es kann beispielsweise weder der One-Point- Crossover noch der Two-Point-Crossover Operator für Permutationen verwendet werden, da die daraus resultierenden Nachkommen in dem meisten Fällen gar keine Permutationen mehr sind. Ein weiteres Problem ist, dass einige der durch Permutationen repräsentierten Lösungen von Optimierungsproblemen mehrere Permutationen haben können, welche äquivalente Lösungen repräsentieren. Ein Beispiel hierfür ist etwa das Travelling Salesman Problem, bei dem eine möglichst kurze Rundreise in einem gegebenen kantenbewerteten Graphen gefunden werden soll. Hier stellt eine Permutation eine Rundreise dar. Allerdings repräsentieren alle Permutationen, die zyklisch verschoben sind, dieselbe Rundreise.

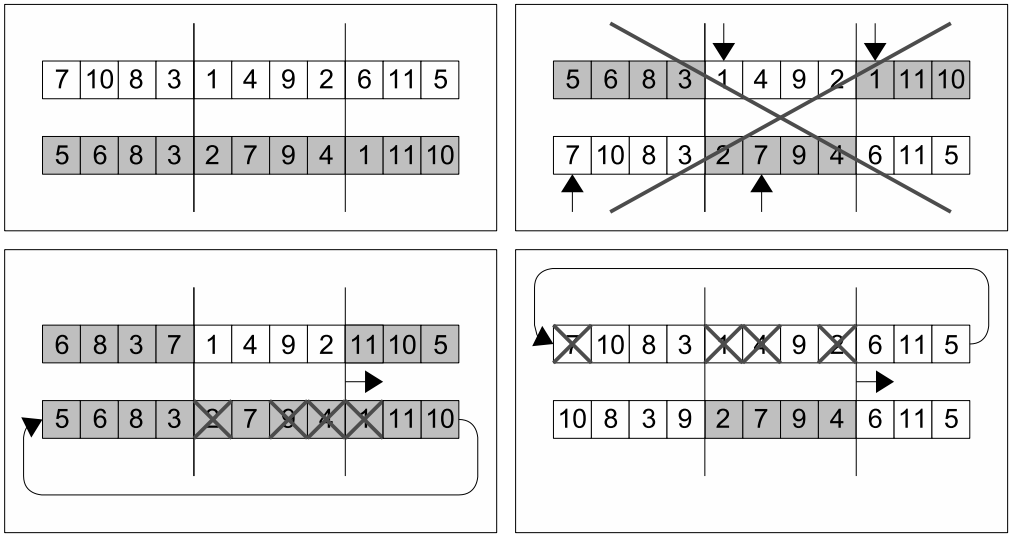


Abbildung 2.5: Crossover von Davis für Permutationen.

Auch ist die rückwärts gelesene Permutation einer Rundreise wieder dieselbe Rundreise. Solche Symmetrien bergen besondere Schwierigkeiten in sich, falls Rekombinationsmechanismen verwendet werden.

Crossover-Operatoren für Permutationen

Es werden an dieser Stelle einige Crossover-Operatoren vorgestellt, die speziell für Lösungen entwickelt wurden, die durch Permutationen repräsentiert werden. Die Darstellung folgt dabei der von Whitley [36].

Davis' Order Crossover Es werden zwei Eltern $A = (a_0, \ldots, a_n)$ und $B = (b_0, \ldots, b_n)$ ausgewählt und zwei Schnittstellen s_0 und $s_1 \geq s_0$ zufällig bestimmt. Nun wird der Teil zwischen den Schnittstellen von A in den Nachkommen C an dieselbe Stelle kopiert, also $c_i = a_i \quad \forall i \in \{s_0, \ldots, s_1\}$. Anschließend wird beginnend beim hinteren Schnittpunkt der Nachkomme mit Elementen aus B aufgefüllt, wobei diejenigen Elemente aus B übersprungen werden, die bereits im Nachkommen vorhanden sind. Ist das Ende einer der Strings erreicht, so wird an Position 0 weitergemacht. Dadurch wird sichergestellt, dass der Nachkomme wieder eine Permutation ist. Eine besondere Eigenschaft dieses Rekombinationsoperators ist, dass der Nachkomme zweier identischer Eltern wieder dasselbe Individuum ist, also $A = B \Rightarrow C = A = B$. Das Verfahren ist in Abbildung 2.5 dargestellt. Oben links im Bild ist sieht man die Ausgangssituation, also die beiden zur Rekombination ausgewählten Eltern A (oben) und B (unten). Im oberen rechten Teil ist dargestellt, was bei einem One-Point-Crossover passieren würde: Der Nachfahre ist keine Permutation mehr. Im Oberen der beiden Nachfahren ist die 1 doppelt, im Unteren die 7 (siehe Pfeile). Im unteren linken Teil des Bildes ist das Ergebnis des Davis' Order Crossover zu sehen. Im unteren String sind zur Veranschaulichung die übersprungenen Elemente aus Elter

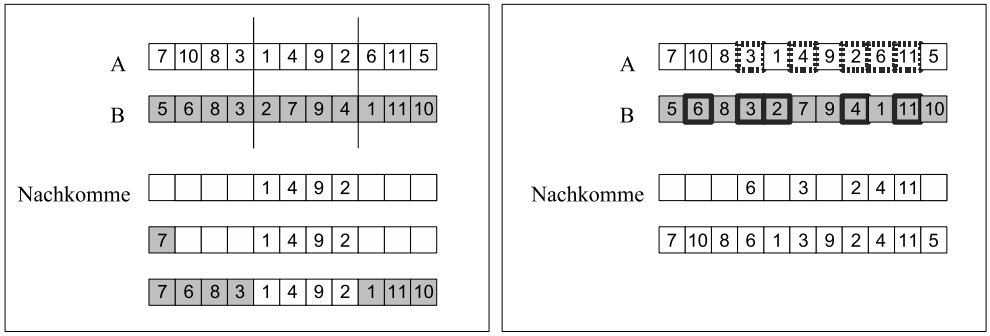


Abbildung 2.6: Links: PMX. Rechts: Order Crossover-2.

B rot durchgestrichen. Wendet man dieselbe Vorgehensweise mit vertauschten Rollen der Eltern an, so erhält man das Ergebnis, was im unteren rechten Bild zu erkennen ist.

Partially Mapped Crossover Der erste Teil diese Crossover-Operators ist identisch zu Davis' Order Crossover: Es werden zwei Eltern ausgewählt, die Schnittstellen bestimmt und der Bereich zwischen den Schnittstellen aus Elter A in den Nachkommen kopiert. Nun geht es jedoch anders weiter: Es werden zunächst alle Elemente aus dem Bereich zwischen den Schnittstellen von Elter B nach dem folgenden Schema verteilt:

- Falls das Element bereits im Nachkommen vorkommt passiert nichts.
- Andernfalls muss das Element b_i noch im Nachkommen platziert werden. Dazu wird zunächst das Element b_j in B gesucht, welches mit dem Element a_i identisch ist. Ist die Position j im Nachkommen noch frei, so wird b_i dorthin kopiert. Ist die Position im Nachkommen nicht frei, so wird nun in B die Position des Elements gesucht, dass im Nachkomme an Position j steht usw., bis eine freie Stelle gefunden wurde. Auf diese Weise können alle Elemente aus Elter B zwischen den Schnittpunkten in den Nachfahren kopiert werden.

Die restlichen Stellen können einfach direkt aus Elter B von derselben Stelle kopiert werden, da es sich genau um die noch fehlenden Elemente handelt.

Order Crossover-2 Es werden zwei Eltern $A = (a_0, \ldots, a_n)$ und $B = (b_0, \ldots, b_n)$ ausgewählt. Zusätzlich werden K Positionen $p_0, \ldots, p_{K-1}$ bestimmt. Nun wird zu jedem b_{p_i} in Elter B das entsprechende Element a_j in A gesucht. Die a_j werden an dieselben Stellen im Nachfahren kopiert, allerdings werden die Elemente so umgeordnet, dass sie dieselbe Reihenfolge wie in Elter B haben. Die verbleibenden Positionen werden aus A übernommen. Es gibt eine Variante des Order Crossover-2 namens *Position Crossover*. Diese führt auf anderem Wege zum selben Ergebnis.

Davis' Uniform Crossover Dieser Operator ist genau genommen derselbe wie der Order Crossover-2 bzw. der Position Crossover. Allerdings werden hier beide möglichen Nachkommen gebildet, indem für den zweiten Nachkommen die Rolle der Eltern vertauscht wird.

Weitere Crossover Operatoren Es wurde eine Reihe weiterer Crossover Operatoren insbesondere für das Traveling Salesman Problem entworfen. Hierzu zählen beispielsweise die Operatoren *Edge recombination*, *Maximal Perservative Crossover (MPX)*, *Cycle Crossover* und *Merge Crossover*. Diese gehen auf die besonderen Anforderungen ein, die entstehen, wenn Permutationen Rundreisen repräsentieren. An dieser Stelle wird nicht weiter auf diese Operatoren eingegangen und stattdessen auf die Literatur [36] verwiesen.

2.3 Graphen

In diesem Abschnitt werden die wichtigsten Grundlagen aus der Graphentheorie dargestellt. Dazu wird zunächst geklärt, was ein Graph ist und was seine Eigenschaften sind. Es folgen einige Grundlagen aus dem Bereich des Graphenzeichnens.

2.3.1 Graphentheorie

Graphen spielen nicht nur in der Informatik eine wichtige Rolle. Die Graphentheorie ist ein gut gediehener Zweig der Mathematik und die große Menge von Veröffentlichungen in diesem Bereich erwschweren den Einstieg, insbesondere weil sogar grundlegende Bezeichnungen im deutschsprachigen Raum nicht einheitlich dargestellt werden. Daher steht an dieser Stelle eine kurze Einführung der wichtigsten Begriffe und Definitionen. Dabei werde in weiten Teilen die Darstellungen von Diestel [10] bzw. Bodendiek und Lang [6] übernommen.

Graphen sind besonders gut geeignet, um Objekte und Beziehungen zwischen diesen Objekten darzustellen. Dies macht sie für viele praktisch relevante Probleme zu einem geeigneten Modell und damit zu einer geeigneten Datenstruktur für die Bearbeitung dieser Probleme durch Computer.

Definition 2.4. *Ein Graph ist ein geordnetes Paar disjunkter Mengen $G = (V, E)$ so, dass gilt $E \subseteq V \times V$. Die Elemente von V werden als* Knoten *bezeichnet. Die Elemente von E heißen* Kanten. *Man unterscheidet zwischen gerichteten und ungerichteten Kanten. Für eine Kante $e \in E$ zwischen zwei Knoten $v_1, v_2 \in V$ schreibt man im Fall einer ungerichteten Kanten $e = \{v_1, v_2\}$ und im Fall einer gerichteten Kanten $e = (v_1, v_2)$.Ein Graph mit gerichteten Kanten heißt* gerichteter Graph *oder* Digraph *(von directed graph). Ist nichts weiter angegeben, so gehen wir vom ungerichteten Fall aus. Ist es erlaubt, dass es zwischen zwei Knoten mehr als eine Kante gibt, so spricht man von einem* Multigraphen.

Falls $V = \emptyset$, so ist zwangsläufig auch $E = \emptyset$. Das Paar $(\emptyset, \emptyset)$ heißt in diesem Fall der leere Graph. *Der leere Graph wird auch abgekürzt geschrieben als $G = \emptyset$. Für zwei Knoten $v_1$und v_2 und eine Kante $e = (v_1, v_2) \in E$ nennt man v_1 und v_2* adjazent *oder* benachbart. *Weiterhin sagt man, der Knoten $v_1$bzw. v_2 ist inzident zu der Kante e.*

Es ist nahe liegend, statt des gesamten Graphen nur bestimmte Teile zu betrachten. Beispielsweise kann nach Teilen eines Graphen gesucht werden, die eine bestimmte Eigenschaft besitzen. Es werden später noch einige solcher Eigenschaften eingeführt. Zunächst soll jedoch der Begriff des Teilgraphen formalisiert werden.

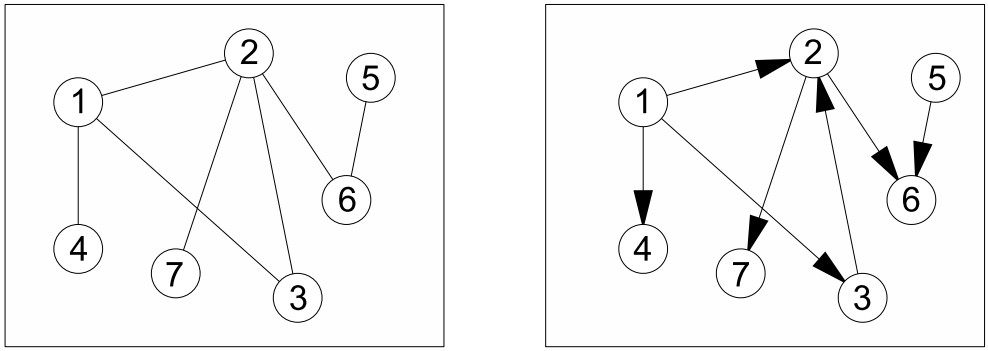


Abbildung 2.7: Links: Ungerichteter Graph. Rechts: Digraph.

Definition 2.5. *Ein Graph $G' = (V', E')$ ist ein Teilgraph von $G = (V, E)$, falls $V' \subset V$ und $E' \subset E$ ist. Man schreibt kurz $G' \subset G$. Ist nur der Graph G und die Menge V' gegeben, so ist der durch V'* induzierte *Teilgraph $(V', E \cap (V' \times V'))$. Man bezeichnet diesen Teilgraphen auch als* Untergraph.

Graphen werden oft zum leichteren Verständnis als Zeichnungen dargestellt. Die in den Lehrbüchern am häufigsten anzutreffende Darstellung ist die, für jeden Knoten einen Punkt zu zeichnen und für jede Kante eine Linie zwischen den Punkten, welche die zu ihr inzidenten Knoten repräsentieren. Um die Identifizierung einzelner Knoten in den Beispielen zu erleichtern, werden diese hier als Kreise dargestellt, in denen sich die zu dem Knoten befindliche Markierung befindet. Zwei einfache Beispiele solcher Graphenzeichnungen sind in Abbildung 2.7 zu sehen. In beiden Beispielen ist $V = \{1, 2, 3, 4, 5, 6, 7\}$. Im linken Bild ist ein ungerichteter Graph mit den Kanten

$$E = \{\{1, 4\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 6\}, \{2, 7\}, \{5, 6\}\}$$

dargestellt. Im rechten Bild ist ein gerichteter Graph mit den Kanten

$$E = \{(1, 4), (1, 2), (1, 3), (2, 5), (2, 6), (2, 7), (3, 2), (5, 6)\}$$

zu sehen. Auf das Zeichnen von Graphen wird in Abschnitt 2.3.2 noch genauer eingegangen.

Gerichtete Kanten werden in den Zeichnungen im Allgemeinen als Pfeile gezeichnet. Für einige Anwendungen ist es notwendig, dass man den Kanten eines Graphen einen Zahlenwert zuordnen kann. Diese Zahlenwerte können beispielsweise Entfernungen oder Kosten darstellen und es somit erlauben, weitere Problemklassen als Graphenprobleme zu behandeln.

Definition 2.6. *Gegeben sei ein Graph $G = (V, E)$, eine Menge M und ein Funktion $\gamma : E \rightarrow M$, die jeder Kante des Graphen eine Objekt aus M zuweist. Dann heißt $G = (V, E, \gamma)$ ein kantengewichteter Graph. Der einer Kante zugeordnete Wert heißt sein* Kantengewicht.

Hat man nun einen Graphen, so kann man in Gedanken über die Kanten von Knoten zu Knoten „wandern“.

Definition 2.7. *Gegeben sei ein Graph $G = (V, E)$. Unter einem* Pfad *P in G versteht man eine geordnete Folge $\langle x_0, x_1, \ldots, x_n \rangle$ von Knoten so, dass $\{x_i, x_{i+1}\} \in E$ für alle $i \in \{0, \ldots, n-1\}$. Die Anzahl n der Kanten, die auf diesem Pfad liegen, nennt man die* Länge des Pfades *P. Für alle Pfade der Länge k scheibt man kurz P^k.*

Beispielsweise ist im ungerichteten Graphen in Abbildung 2.7 $1, 2, 6, 5$ ein Pfad der Länge 3. Bei gerichteten Graphen muss natürlich die Kantenrichtung beachtet werden, so dass es im gerichteten Beispielgraphen keinen solchen Pfad gibt, da es keine Kante von Knoten 6 zu Knoten 5 gibt.

Betrachtet man nun, welche Knoten eines Graphen jeweils durch Pfade miteinander verbunden sind, so gelangt man zum Begriff des Zusammenhangs.

Definition 2.8. *Existiert in einem Graphen $G = (V, E)$ ein Pfad zwischen zwei Knoten $v_1, v_2 \in V$, so sagt man, diese Knoten sind* zusammenhängend. *Die maximalen Teilmengen $Z_i \subseteq V$ paarweise zusammenhängender Knoten heißen die (einfachen)* Zusammenhangskomponenten von *G.*

Neben den einfachen Zusammenhangskomponenten kann man auch solche Zusammenhangskomponenten betrachten, die zweifach zusammenhängen, die also nach Herausnahme eines beliebigen Knotens und aller dazu inzidenten Kanten aus E immer noch zusammenhängend sind.

Definition 2.9. *Gegeben sein ein Graph $G = (V, E)$.*

- *Falls für alle $v_1 \in V$ der durch $V \backslash v_1$ induzierte Untergraph von V einfach zusammenhängend ist, so heißt G* zweifach zusammenhängend. *Die maximalen zweifach zusammenhängenden Komponenten von V heißen die* Zweizusammenhangskomponenten *von V.*
- *Falls für alle $v_1, v_2 \in V$ der durch $V \backslash \{v_1, v_2\}$ induzierte Untergraph V einfach zusammenhängend ist, so heißt G* dreifach zusammenhängend. *Die maximalen dreifach zusammenhängenden Komponenten von V heißen die* Dreizusammenhangskomponenten *von V.*

Wenn ein Graph durch eine Zeichnung dargestellt werden soll, wird im Allgemeinen versucht, die Zeichnung so klar wie möglich zu machen. Ein wichtiger Aspekt dabei ist, das die Kreuzung von Kanten vermieden wird.

Definition 2.10. *Eine* Graphenzeichnung Π *(kurz: Zeichnung) zu einem gegebenen Graphen $G = (V, E)$ ist eine Abbildung mit den folgenden Eigenschaften:*

1. *Jedem Knoten $v \in V$ wird ein eindeutiger Punkt $\Pi(v)$ der Ebene $\mathbb{R}^2$ zugeordnet.*
2. *Jeder Kante $(v_1, v_2) \in E$ wird eine Kurve in der Ebene $\mathbb{R}^2$ so zugeordnet, dass die Endpunkte der Kurve den Bildern der zugehörigen Knoten $\Pi(v_1)$ und $\Pi(v_2)$ entsprechen.*

Die Graphenzeichnung heißt planar, *wenn das Innere keiner Kurve zugleich Teil einer anderen Kurve ist. Sofern keine Verwechselungsgefahr besteht, spricht man auch kurz von den* Knoten der Zeichnung *und den* Kanten der Zeichnung, *lässt also den Zusatz „Abbild von" weg.*

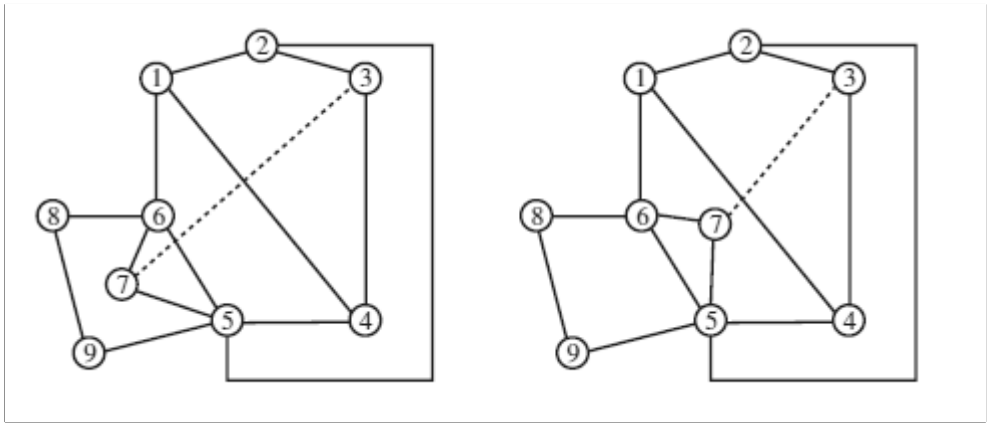


Abbildung 2.8: Zwei verschiedene kombinatorische Einbettungen eines Graphen. Beim Einfügen der Kante $\{3, 7\}$ werden unterschiedlich viele neue Kreuzungen erzeugt [16].

Die Kanten solcher planaren Graphenzeichnungen teilen die Ebene $\mathbb{R}^2$ *in Regionen ein, die als* Face *bezeichnet werden. Die unendliche Region „außerhalb des Graphen" wird als* äußeres Face *bezeichnet, die anderen sind die* inneren Faces.

Man kann für jeden Graphen eine Zeichnung finden, auf die er isomorph abgebildet werden kann. Einige Graphen können sogar auf planare Zeichnungen isomorph abgebildet werden. Man sagt, der Graph kann *in die Ebene eingebettet* werden.

Definition 2.11. *Gibt es zu einem Graphen eine planare Graphenzeichnung, so nennt man ihn* planar.

Die Anzahl planarer Zeichnungen, die für einen planaren Graphen gefunden werden kann, ist unendlich groß. Viele dieser Zeichnungen sind sich jedoch „in gewisser Weise" ähnlich:

Definition 2.12. *Die* kombinatorischen Einbettungen *eines planaren Graphen sind die Äquivalenzklassen seiner planaren Graphenzeichnungen, bei denen jeweils die inzidenten Kanten aller Knoten im Uhrzeigersinn betrachtet dieselbe Reihenfolge haben.*

Bei der kombinatorischen Einbettung wird also nicht mehr darauf geachtet, wie die Koordinaten der Knoten sind, oder wie die Kanten verlaufen und welche Länge sie haben, lediglich die relative Positionierung von Knoten und Kanten wird betrachtet. Die Wahl der kombinatorischen Einbettungen hat direkten Einfluss auf die Anzahl der notwendigen Kreuzungen beim Einfügen einer Kante. Dies ist in dem Beispiel in Abbildung 2.8 veranschaulicht.

Man kann zu jedem ebenen Graphen einen sogenannten *dualen Graphen* erzeugen. Dazu wird für jedes Face des ebenen Graphen ein Knoten in seinem dualen Graphen erzeugt und je zwei benachbarte Faces des ebenen Graphen erhalten eine Kante zwischen den korrespondierenden Knoten im dualen Graphen. Ein Beispiel für einen ebenen Graphen und seinen dualen Graphen ist im Abbildung 2.9 dargestellt. Da ein Face über mehrere Kanten mit einem anderem Face benachbart sein kann, ist der duale Graph im Allgemeinen offensichtlich ein Multigraph.

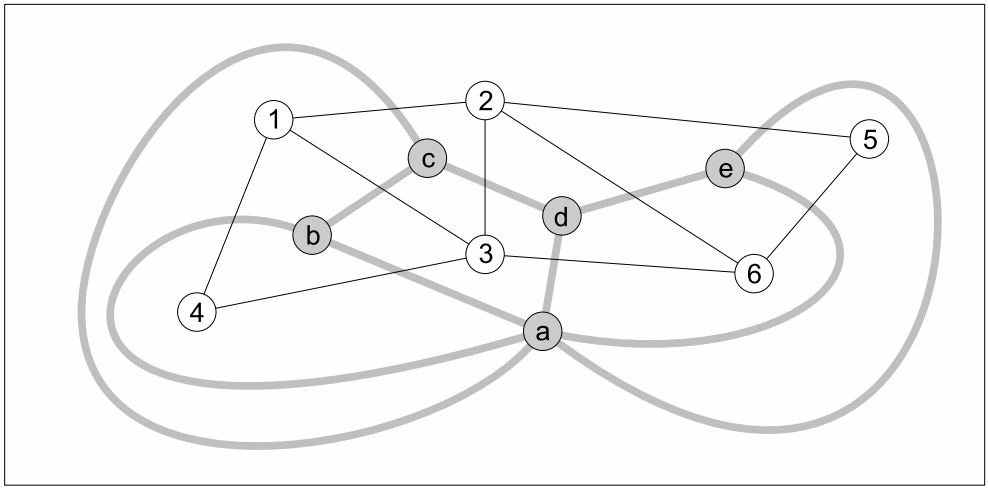


Abbildung 2.9: Ein ebener Graph mit dazugehörigem dualen Graphen.

2.3.2 Zeichnen von Graphen

Dieser Abschnitt beschäftigt sich mit dem Zeichnen von Graphen. In Abschnitt 2.3.1 wurde ja bereits beschrieben, wie diese in Lehrbüchern gezeichnet werden. Hier sollen nun die Ziele, die beim Zeichnen von Graphen verfolgt werden erläutert werden. Dazu werden zunächst einige Anforderungen beschrieben, die an eine gute Graphenzeichnung gestellt werden können. Anschließend wird das *TSM-Modell* beschrieben, das ein viel verwendetes Verfahren ist, um nicht-planare Graphen automatisch zu zeichnen.

Anforderungen an eine Graphenzeichnung

Man kann Graphen unter verschiedenen Gesichtspunkten darstellen. Je nachdem, welches Ziel bei der Erstellung einer Zeichnung verfolgt wird, werden die einzelnen Kriterien verschieden gewichtet. Einige solcher Kriterien sind:

1. Anzahl sich kreuzender Kanten
2. Symmetrie der Zeichnung
3. Anzahl der Kanten-Knicke
4. Winkel zwischen sich kreuzenden Kanten
5. Benötigte Zeichenfläche
6. Kantenlängen
7. Minimaler Knoten- und Kantenabstand
8. Gleichmäßige Nutzung der Zeichenfläche

Die Anzahl der sich kreuzenden Kanten ist das Kriterium, welches die zentrale Rolle im Rest der Arbeit spielt. Statt *sich kreuzender Kanten* wird auch kürzer von *Kreuzungen* gesprochen. Je mehr Kanten sich kreuzend, desto schwerer wird es für den Betrachter, die Zeichnung zu interpretieren. Jede Kreuzung bietet dem Auge beim Verfolgen einer Kante die Möglichkeit, vom verfolgten Pfad abzuschweifen. Die Wahrscheinlichkeit, die falsche Kante nach der Kreuzung zu verfolgen ist dabei umso größer, je kleiner der *Winkel zwischen den sich kreuzenden Kanten* ist. Im Extremfall werden deshalb beim sogenannten *orthogonalen Layout* nur senkrechte Kantenkreuzungen zugelassen. In manchen Fällen ist die Größe der Zeichenfläche vorgegeben, beispielsweise, wenn ein bestimmter Graph auf eine Buchseite gedruckt werden soll. In anderen Fällen soll die benötigte Fläche für die Darstellung des Graphen minimiert werden, wie zum Beispiel beim Platinenentwurf, bei dem die elektronischen Bauteile als spezielle Knoten und die Leiterbahnen als Kanten interpretiert werden können. Ohne Größenminimierung währen Mobiltelefone in der heutigen Größenordnung nicht denkbar. Auch eine Beschränkung der Kantenlänge kann notwendig sein. So wächst der ohmsche Widerstand einer Leiterbahn mit seiner Länge. Auch die Interpretation einer Zeichnung wird durch lange Kanten schwieriger. Sehr kurze Kanten kann das Auge „auf einen Blick" wahrnehmen, während für längere Kanten das Auge dem Verlauf folgen muss. Sind die Kanten zu dicht beieinander, kann das Auge beim Verfolgen versehentlich auf die falsche Bahn überspringen, was zu falschen Interpretationen führt. In dem Platinenbeispiel wirken parallele Leiterbahnen als Kondensator, der insbesondere in der Hochfrequenztechnik zu einem unerwünschten Filtereffekt führt, der eine Begrenzung der maximalen Frequenz zur Folge hat. Dies ist beispielsweise der Grund dafür, dass die Strukturen der Mikrochips immer kleiner werden, denn der Filtereffekt wächst mit der Leiterbahnlänge. Die Forderung nach der gleichmäßigen Nutzung der Zeichenfläche kann einerseits ästhetisch bedingt sein, andererseits aber auch zur besseren Interpretierbarkeit der Zeichnung beitragen. Gibt es Ungleichgewichte in der Nutzung der Zeichenfläche, so können unter Umständen die Schwerpunkte „entzerrt" werden, indem sie eine größere Teilfläche zugewiesen bekommen, so dass sie leichter zu lesen sind. Es ist leicht nachvollziehbar, dass die genannten Kriterien nicht unabhängig voneinander realisiert werden können. Einige der Kriterien bedingen andere, einige stehen zu anderen im Widerspruch. Es gilt daher bei jeder Zeichnung aufs Neue einen für den Zweck geeigneten „Trade-off" zu finden, der den Bedürfnissen des Anwenders entgegen kommt.

Purchase, Cohen und James habe die Auswirkung der Symmetrie, der Anzahl sich kreuzender Kanten und der Anzahl geknickter Kanten von Graphenzeichnungen auf das Verständnis durch Testpersonen untersucht. Für die Symmetrie konnte dabei erstaunlicherweise keine eindeutige Auswirkung nachgewiesen werden, während eine Minimierung der anderen beiden Eigenschaften die Verständlichkeit der Zeichnung deutlich steigern konnte [31].

In einer weiteren Studie konnte Purchase feststellen, dass eine Minimierung der Anzahl sich kreuzender Kanten mit Abstand das wichtigste Kriterium darstellt, um die Lesbarkeit der Zeichnung zu maximieren. Bei den Schnittwinkeln der Kanten und der orthogonalen Darstellung war keine signifikante Verbesserung zu erkennen [30].

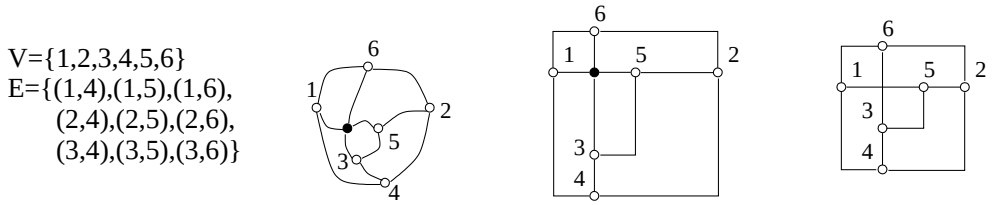


Abbildung 2.10: Topology-Shape-Metric Verfahren.
Von links nach rechts: Graph als abstrakte Beschreibung – Planare Einbettung durch Planarisierung – Orthogonale Darstellung nach Orthogonalisierung – Endgültige Zeichnung nach Kompaktierung. [29]

2.3.3 Das TSM-Modell

Viele Ansätze zum automatischen Zeichnen von Graphen folgen demselben Schema. Es wird in drei Schritten vorgegangen:

1. *Topology.* Im ersten Schritt wird die Topologie festgelegt. Das bedeutet, dass der Graph in die Ebene eingebettet wird. Dies ist für planare Graphen kein Problem. Alle anderen Graphen müssen zuvor *planarisiert* werden, also durch geeignete Methoden planar gemacht werden. Planarisierung wird in Abschnitt 2.4.2 genauer beschrieben.

2. *Shape.* Die Anordnung der Knoten zueinander sowie der Verlauf der Kanten wird festgelegt. Bei einer orthogonalen Anordnung beispielsweise verlaufen die Kanten nun bereits orthogonal, sind aber noch nicht zwangsläufig an ihrem endgültigen Platz.

3. *Metric.* In diesem Schritt erhält die Zeichnung ihre endgültigen Ausmaße. Es werden die absoluten Positionen der Knoten und der Verlauf der Kanten festgelegt.

Dieses Topology–Shape–Metric Modell, kurz als *TSM-Modell* bezeichnet, ist in Abbildung 2.10 dargestellt. Im linken Teil der Abbildung ist der Graph als abstrakte mathematische Beschreibung angegeben. Aus dieser soll automatisch eine ansprechende Zeichnung generiert werden. Im nächsten Teilbild ist bereits die Topologie des Graphen für die Zeichnung festgelegt worden. Gut zu erkennen ist der schwarz dargestellte Dummy-Knoten aus dem Planarisierungsschritt. In der vorletzten Abbildung hat die Zeichnung bereits ihre endgültige Form, jedoch nicht die endgültigen Ausmaße. Offensichtlich wurde hier ein orthogonales Layout gewählt. Im letzten Teil der Abbildung ist als Endergebnis die fertige Graphenzeichnung zu sehen.

Es sei der Vollständigkeit halber an dieser Stelle kurz angemerkt, dass es neben dem TSM-Modell auch noch andere Vorgehensweisen um Graphen automatisch zu zeichnen. Ein Beispiel sind die kräftebasierten Verfahren wie etwa die *Spring Embedder*.

2.4 Kreuzungsminimierung

Wie wir im Abschnitt 2.3.2 gesehen haben, ist ein Merkmal für die Qualität einer Graphenzeichnung die Anzahl der sich kreuzenden Kanten. In diesem Abschnitt soll die Minimierung dieser Anzahl etwas genauer betrachtet werden.

2.4.1 Das Kreuzungsminimierungsproblem

Definition 2.13. *Das Kreuzungsminimierungsproblem ist das Problem zu einem gegebenen Graphen $G = (V, E)$ eine Zeichnung zu finden, bei der die Anzahl gekreuzter Kanten minimal ist. Das Kreuzungsminimierungsproblem ist also ein Optimierungsproblem $(\mathcal{S}, f)$, bei dem der Suchraum $\mathcal{S}$ die Menge aller möglichen Planarisierungen von G ist und die Zielfunktion f jeder Planarisierung die Anzahl notwendiger Kreuzungen ihrer Zeichnung zuordnet.*

Die Anzahl der Kreuzungen kann nicht beliebig klein werden.

Definition 2.14. *Die* Kreuzungszahl $cr(G)$ *ist die kleinste Anzahl von Kreuzungen aller Zeichnungen von G.*

Offensichtlich ist die Kreuzungszahl planarer Graphen 0. Johnson und Johnson [20] haben gezeigt, dass die Berechnung der Kreuzungszahl im Allgemeinen NP hart ist.

2.4.2 Planarisierung

Der erste Schritt des TSM-Modells sieht vor, nicht planarer Graphen in planare Graphen umzuwandeln, damit diese in die Ebene eingebettet werden können. Dieser Schritt wird als Planarisierung bezeichnet. Die Planarisierung wird in der Praxis dadurch erreicht, dass in den nicht-planaren Graphen zusätzliche Knoten eingefügt werden, die anstelle einer Kreuzung stehen. Um geeignete Stellen für das Einfügen solcher *Dummy-Knoten* zu finden, arbeitet ein Planarisierungsalgorithmus in zwei Schritten:

1. Entfernen einer möglichst kleinen Menge von Kanten, so dass der resultierende Graph planar wird.
2. Wiedereinfügen der Kanten, so dass möglichst wenige Kreuzungen entstehen. Anstelle der Kreuzungen werden Dummy-Knoten eingefügt.

Jeder dieser beiden Schritte ist für sich genommen NP hart. Beim Wiedereinfügen der Kanten im zweiten Schritt werden zwangsläufig neue Kreuzungen erzeugt, die zu Dummy-Knoten werden. Wie viele solcher Dummy-Knoten erzeugt werden, hängt einerseits davon ab, welche Kanten zuvor entfernt wurden, um Planarität zu erreichen, andererseits von dem verwendeten Algorithmus zum Wiedereinfügen der Kanten. In der Regel werden die Kanten sukzessive eingefügt, also jeweils nur die nächste Kante aus der Liste aller wiedereinzufügenden Kanten. Bei dieser Vorgehensweise ist die Reihenfolge, in welcher die Kanten eingefügt werden, ebenfalls ausschlaggebend für die Anzahl daraus resultierender Kreuzungen.

In den nächsten beiden Abschnitten werden zwei Algorithmen für das kreuzungsminimale Einfügen einer Kante in einen gegebenen planaren Graphen vorgestellt. Der erste dieser beiden Algorithmen verändert die kombinatorische Einbettung des übergebenen Graphen nicht. Man spricht vom Einfügen einer Kante bei *fixer kombinatorischer Einbettung*. Der zweite Algorithmus ist etwas raffinierter. Er fügt die Kante so in den Graphen ein, dass unter allen kombinatorischen Einbettungen diejenige ausgewählt wird, die zu der geringsten Anzahl von Kreuzungen führt. Dieses Vorgehen wird auch als Einfügen einer Kante bei *variabler kombinatorischer Einbettung* bezeichnet.

2.4.3 Einfügen einer Kante bei fixer Einbettung

Soll eine Kante in eine vorgegebene kombinatorische Einbettung kreuzungsminimal eingefügt werden, so kann dies effizient mit Hilfe des dualen Graphen geschehen. Es wird also zu der kombinatorischen Einbettung zunächst der duale Graph gebildet.

Gegeben sein ein Graph $G = (V, E)$, eine kombinatorische Einbettung Π sowie zwei Knoten $v_1, v_2 \in V$ zwischen denen eine Kante kreuzungsminimal eingefügt werden soll. Die Kante wird wie folgt in den Graphen eingefügt: Die Methode zum Einfügen der Kante in den Graphen ist in Abbildung 2.11 dargestellt:

1. Im ersten Bild ist der Graph in der kombinatorischen Einbettung dargestellt, in der die Kante eingefügt werden soll. Die Endknoten der einzufügenden Kante $1, 6$ sind hervorgehoben. Der schraffierte Bereich repräsentiert einen ausreichend großen, dreifach zusammenhängenden Teil des Graphen, der für das Beispiel sicherstellt, dass es nur eine Möglichkeit gibt, die Kante kreuzungsminimal einzufügen.

2. Es wird zunächst der duale Graph erzeugt. In der Darstellung ist dies der grau dargestellte Multigraph. Für das Beispiel irrelevante Teile des dualen Graphen wurden zur besseren Überschaubarkeit ausgelassen.

3. Der duale Graph wird nun erweitert. Der Start- und der Endknoten werden zu dem dualen Graphen hinzugefügt. Weiterhin werden Kanten zu allen Knoten erzeugt, die benachbarte Faces im ursprünglichen Graphen repräsentieren.

4. In dem erweiterten dualen Graphen wird nun der kürzeste Weg zwischen den beiden Endknoten der neuen Kante gesucht. Dieser wird die neue Kante werden. Es werden dabei so viele neue Kreuzungen erzeugt, wie der gefundene Weg lang ist, abzüglich der ersten und letzten Kante des Weges, die ja erst durch die Erweiterung hinzugekommen ist. Dies ist leicht einzusehen, da eine Kante im dualen Graphen ja für benachbarte Faces im ursprünglichen Graphen steht, diese also durch genau eine Kante getrennt sind, welche die neu hinzukommende Kante kreuzt.

5. Letztendlich werden alle hinzugekommenen Kreuzungen durch Dummy-Knoten ersetzt.

6. Das Ergebnis des Kanteneinfügens ist wieder ein planarer Graph, auf dem sukzessive mit demselben Algorithmus weitere Kanten eingefügt werden können.

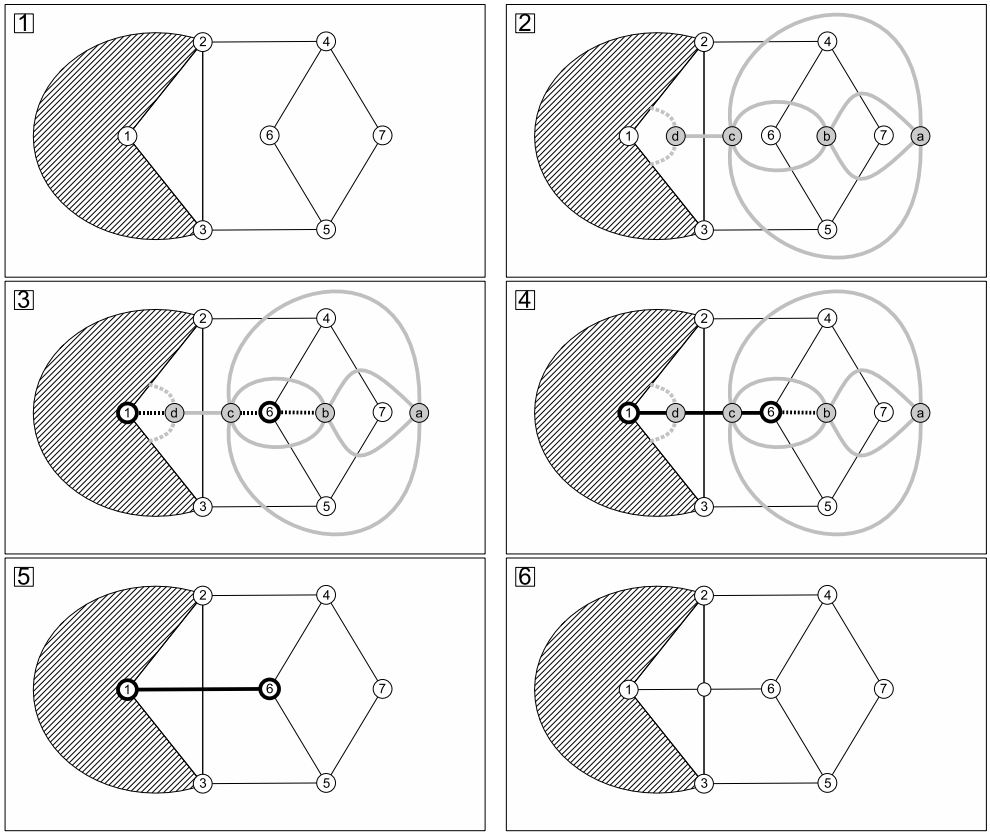


Abbildung 2.11: Vorgehen beim Einfügen einer Kante bei fixer kombinatorischer Reihenfolge.

2.4.4 Einfügen einer Kante bei variabler Einbettung

Neben dem Algorithmus zum Einfügen einer Kante bei gegebener kombinatorischer Einbettung gibt es auch einen Algorithmus zum optimalen Einfügen einer Kante in einen planaren Graphen. Der Algorithmus ist noch recht jung und wurde von Gutwenger, Mutzel und Weiskircher vorgestellt [16].

Der Algorithmus verwendet die Datenstruktur *SPQR-Tree*, die es ermöglicht, alle kombinatorischen Einbettungen gleichzeitig zu bearbeiten. Gutwenger und Mutzel haben bereits zuvor gezeigt, dass SPQR-Trees in linearer Zeit berechnet werden können [14]. Dies führt zu einem effizienten Algorithmus für das optimale Einfügen einer Kante. Allerdings führt das wiederholte optimale Einfügen einzelner Kanten im Allgemeinen nicht zu der kleinsten Anzahl von Kreuzungen bei der Planarisierung. Auch dieser Algorithmus basiert, ähnliche dem Algorithmus zum Einfügen einer Kante bei fixer Einbettung, auf der Suche nach kürzesten Wegen.

2.5 Experimente

Dieser Abschnitt widmet sich den Grundlagen von Computerexperimenten. Zunächst werden in Abschnitt 2.5.1 einige Grundlagen aus der Statistik eingeführt, da die Analyse von Experimenten ohne statistische Hilfsmittel nicht effektiv möglich ist. Daran anschließend wird die Vorgehensweise beim Experimentieren nach der Methode „Design of Experiments" (DOE) beschrieben. Diese wurde ursprünglich für die Durchführung von Experimenten im industriellen Umfeld entwickelt, ihre Methoden finden aber mittlerweile auch Anwendung bei Computerexperimenten. Der darauf folgende Teil behandelt die Methode „Design and Analysis of Computer Experiments" (DACE), die speziell für die Durchführung von Experimenten entwickelt wurde, die auf dem Computer simulierten werden. Das besondere an diesen Experimenten ist, dass sie auf deterministischen Funktionen basieren. Zum Abschluss wird die Methode „Serial Parameter Optimization" vorgestellt, die DOE und DACE geschickt kombiniert, um sie für die Optimierung der Parameter von nichtdeterministischen evolutionären Algorithmen nutzen zu können.

2.5.1 Statistik

Dieser Abschnitt gliedert sich in zwei Teile. In der ersten Hälfte werden statistische Grundbegriffe eingeführt, die für das Verständnis der folgenden Abschnitte hilfreich sind. Der zweite Teil stellt einige graphische Hilfsmittel vor, die in der Statistik oft anzutreffen sind.

Grundbegriffe

Um sicherzustellen, dass alle verwendeten statistischen Begriffe bekannt sind, gibt dieser Abschnitt einen kurzen Überblick. Für eine ausführliche Einleitung sei auf die einführende Literatur verwiesen, wie beispielsweise das Lehrbuch von Spiegel und Stephens [34].

Grundgesamtheit Die Grundgesamtheit bezeichnet eine Sammlung von Daten, die ein bestimmtes Merkmal repräsentieren. Ein Beispiel wäre das Alter aller Studenten der Universität Dortmund.

Stichprobe Unter einer Stichprobe versteht man einen Teil der Grundgesamtheit. Das Teilgebiet der Statistik, das als *induktive Statistik* bezeichnet wird, untersucht, unter welchen Umständen von einer Stichprobe auf die Grundgesamtheit geschlossen werden kann. In dem Beispiel des Alters der Studierenden wäre die Frage, wie groß die Stichprobe sein muss, um einigermaßen sichere Aussagen über die Altersverteilung an der ganzen Universität machen zu können, Gegenstand einer solchen Untersuchung.

Median Ordnet man eine Datenmenge der Größe nach an, so ist der Median das mittlere Element. Falls es einen gerade Anzahl von Daten gibt, so wird der Median als der arithmetische Mittelwert der beiden Elemente in der Mitte berechnet. Der Median teilt die Datenmenge also in zwei gleichgroße Bereiche.

Quartile Die Quartile können als eine Verfeinerung des Medians aufgefasst werden. Teilt man eine der Größe nach sortierte Datenmenge in vier Teile, so erhält man die drei Quartile Q_1, Q_2 und Q_3, wobei die mittlere Quartile gleich dem Median ist. In ähnlicher Weise kann man die Menge in 10 Teile aufteilen und erhält damit die *Dezile* $D_1, \ldots, D_9$ oder man Teilt die Menge in 100 Teile und erhält damit die *Perzentile* $P_1, \ldots, P_{99}$.

Abweichung Unter der Abweichung wird der Grad verstanden, zu dem numerische Daten um einen Durchschnittswert verteilt sind. Es gibt verschiedene Maße für die Abweichung.

Variationsbreite Die Variationsbreite einer Menge von Daten ist die Differenz von größtem und kleinstem Element der Menge.

Arithmetisches Mittel Das arithmetische Mittel $\overline{X}$ von N Zahlen $X_1, \ldots, X_N$ ist definiert als

$$\overline{X} = \frac{X_1 + \ldots + X_N}{N}.$$

Mittlere Abweichung Die mittlere Abweichung MD von N Zahlen $X_1, \ldots, X_N$ kann berechnet werden als

$$\mathrm{MD} = \frac{\sum_{j=1}^{N} \left| X_j - \overline{X} \right|}{N}.$$

Standardabweichung Die Standardabweichung s einer Menge von N Zahlen $X_1, \ldots, X_N$ wird berechnet durch

$$s = \sqrt{\frac{\sum_{j=1}^{N} \left(X_j - \overline{X} \right)^2}{N}}.$$

Varianz Die Varianz S^2 ist das Quadrat der Standardabweichung.

Hypothese Unter einer Hypothese versteht man eine Behauptung, die vor einem Experiment aufgestellt wird und die durch das Experiment getestet werden soll. Das Experiment liefert Daten, die Hinweise für die Korrektheit der Hypothese geben. Hypothesen sind entweder wahr oder falsch. Sie können aufgrund der vorliegenden Daten akzeptiert oder verworfen werden. Ein Spezialfall ist die *Nullhypothese*. Darunter versteht man eine Hypothese, die aufgestellt wird mit dem Ziel, sie zu verwerfen.

Fehler 1. Art Ein Fehler der 1. Art liegt vor, wenn eine wahre Hypothese verworfen wird.

Fehler 2. Art Ein Fehler der 2. Art liegt vor, wenn eine falsche Hypothese akzeptiert wird.

Signifikanzniveau Die höchste Wahrscheinlichkeit, bei der Überprüfung einer Hypothese einen Fehler 1. Art zu begehen, wird als Signifikanzniveau bezeichnet. Oft anzutreffende Signifikanzniveaus sind beispielsweise $0,01$ und $0,05$.

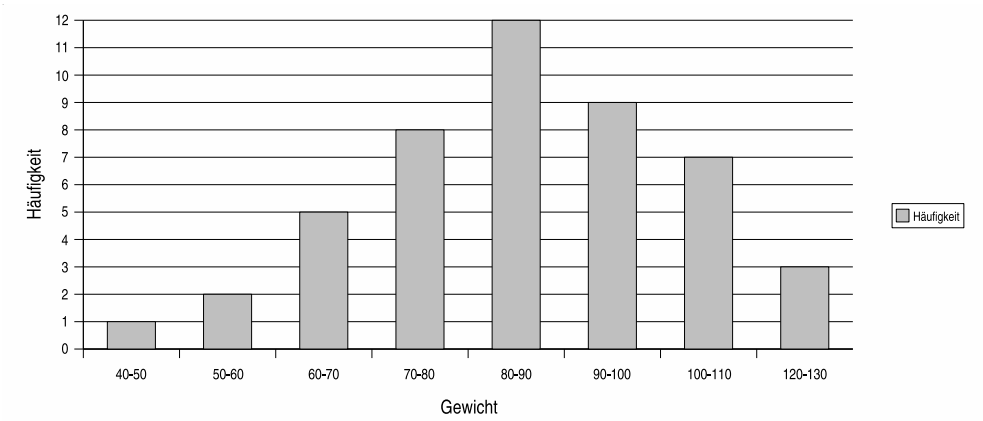


Abbildung 2.12: Ein Histogramm

Graphische Hilfsmittel

Bei statistischen Analysen sind häufig sehr große Datenmengen Gegenstand der Betrachtung. Eine Darstellung in Tabellenform ist dabei oft nicht geeignet, um Zusammenhänge zu erklären. Es haben sich eine Reihe von Darstellungsarten in der Statistik etabliert, von denen einige hier dargestellt werden.

Soll die Verteilung von Daten dargestellt werden, so ist das *Histogramm* eine geeignete Form. Bei einem Histogramm werden die Daten zu Gruppen zusammengefasst. Liegen beispielsweise kontinuierliche Daten vor, so kann der Datenbereich in eine Menge gleichgroßer Intervalle partitioniert werden. Für jedes dieser Intervalle wird nun die Anzahl darin vorkommender Datenpunkte gezählt. Diese Anzahl bestimmt die Länge eines Balkens, der im Diagramm die Anzahl der Datenpunkte darstellt. Ein Beispiel für ein Histogramm ist in Abbildung 2.12 dargestellt. Das Diagramm stellt die Anzahl Personen einer Stichprobe dar, die in eine bestimmte Gewichtskategorie fallen. Das größte Problem bei Histogrammen ist, dass über die Datenpunkte eines Intervalls keine Detailinformationen erhalten bleiben. Es eignet sich daher insbesondere, um einen Überblick über die Verteilung von gesammelten Daten zu gewinnen.

Eine andere Möglichkeit einen Überblick über die Daten zu bekommen sind die sogenannten *Boxplots*. Diese stellen das Minimum, das Maximum, die obere und untere Quartile sowie den Median dar. Ein Beispiel für einen Boxplot befindet sich in Abbildung 2.13. Der dargestellte Kasten erstreckt sich von der oberen zur unteren Quartile. Der Median ist als Linie innerhalb des Kastens zu erkennen. An den Enden des Kastens befinden sich weitere Linien, die sich bis zum Minimum bzw. Maximum des Datenbereichs erstrecken. Boxplots geben eine sehr gute Übersicht, in welchem Bereich sich die Daten einer Stichprobe konzentrieren. Insbesondere wenn die Daten mehrerer Versuchsreihen als Boxplots in einem Diagramm dargestellt werden, kann schnell erkannt werden, ob einer der Versuche bessere Ergebnisse erzielen konnte und ob diese konzentriert oder gestreut liegen.

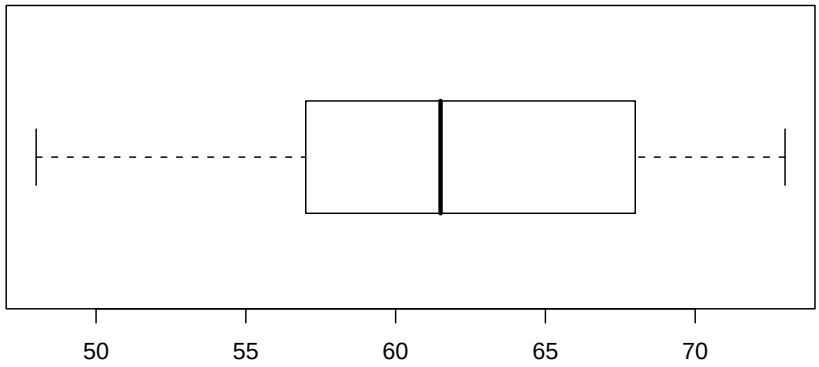


Abbildung 2.13: Ein Boxplot. Der Median ist als breite Linie innerhalb des Kastens dargestellt.

2.5.2 DOE

Die „klassischen" Techniken zum Aufstellen und Auswerten experimenteller Designs, wie sie etwa in der Industrie Verwendung finden, werden unter dem Begriff *Design of Experiments* (DOE) zusammengefasst. Einige Begriffe aus diesem Bereich werden in diesem Abschnitt kurz erläutert, ohne dabei zu sehr in die Tiefe zu gehen, da die Durchführung von DOE von Hand nicht Teil dieser Arbeit ist. Allerdings wird in den Experimenten DOE implizit in den verwendeten Verfahren verwendet, so dass eine kurze Einführung sinnvoll erscheint. Weiterführende Erläuterungen der Verfahren findet man bei Box, Box und Hunter [8] oder bei Montgomery [27].

Grundbegriffe

Experiment Nach Montgomery ist ein Experiment eine Serie von Tests, bei der kontrollierte Veränderungen an den Eingangsgrößen des betrachteten Systems vorgenommen werden, um Veränderungen an den Ausgangsgrößen erkennen und begründen zu können [27].

Faktor Die Faktoren eines Experiments sind die Eingangsgrößen des betrachteten Systems, die auf den Verlauf des Experiments Einfluss nehmen. Man unterscheidet zwischen *kontrollierbaren Faktoren* und *unkontrollierbaren Faktoren*.

Level Die verschiedenen Werte, die ein Faktor im Verlauf eines Experiments annehmen kann, werden als Level bezeichnet.

Responsevariable Die bei einem Experiment betrachteten Ausgangsvariablen des Systems werden als Responsevariable bezeichnet.

2.5.2.1 Grundlegende Prinzipien des Experimentierens

Experimente sollen im Allgemeinen möglichst effizient durchgeführt werden. Dazu bedarf es eines wissenschaftlichen Ansatzes, um das Experiment zu planen und auszuwerten. Wenn das im Experiment untersuchte Problem Daten beinhaltet, die einem experimentellen Fehler unterliegen, dann ist eine statistische Analyse der einzige Weg, objektiv Schlussfolgerungen aus dem Experiment zu ziehen. Die Planung eines Experiments mit dem Ziel, geeignete Daten zu sammeln, die mit statistischen Methoden analysiert werden, heißt *statistisches Design von Experimenten*. Für ein solches Design gibt es drei grundlegende Prinzipien. Die folgende Aufzählung erläutert diese:

1. **Wiederholung**
   Mit „Wiederholung" ist die mehrfache Durchführung des Experiments gemeint. Durch Wiederholungen kann einerseits der experimentelle Fehler abgeschätzt werden. Andererseits kann der Effekt eines Faktors genauer ermittelt werden.

2. **Blocking**
   Das Blocking ist eine Technik um die Genauigkeit eines Experiments zu erhöhen. Ein *Block* ist ein Teil des Experiments mit ähnlichen Eigenschaften. Soll beispielsweise die reale Laufzeit eines Algorithmus untersucht werden, so würde man jeweils *Blocks* mit ähnlichen Eingabedaten auf jeweils demselben Rechner berechnen, auch wenn mehrere vermeintlich identische Rechner zur Verfügung stehen. Einflüsse, die auf nicht absolut identische Bauteile in den Rechnern zurückzuführen sind, werden dadurch vermieden.

3. **Randomisierung**
   Mit Hilfe von Randomisierung, also der Einbeziehung des Zufalls in die Planung eines Experiments, können zwei Dinge beeinflusst werden:

   - Die zufällig Auswahl des Materials eines Experiments.
   - Die zufällige Reihenfolge, mit der die Versuche eines Experiments durchgeführt werden.

   Die statistische Auswertung beruht auf der Annahme, das Beobachtungen sowie Fehler unabhängige voneinander sind. Dies wird durch korrekte Randomisierung sichergestellt.

Richtlinien für das Design von Experimenten

Montgomery schlägt folgendes Vorgehen für das Design und die Analyse von Experimenten vor [27, S. 14ff]:

E-1 *Erkennen und Beschreiben des Problems*
So offensichtlich dieser Punkt erscheint, so schwierig ist es oft in der Praxis, ein Problem, das eines Experiments bedarf, zu identifizieren bzw. eine genaue Beschreibung aufzustellen. Bei einer industriellen Anlage müssen beispielsweise alle betreffenden Parteien befragt werden, also die Ingenieure, das Management, die Qualitätssicherung usw., um eine genaue Beschreibung des Problems überhaupt aufstellen zu können. Das Design des Experiments wird dadurch zu einer Teamaufgabe.

E-2 *Auswahl der Faktoren, Level und Wertebereiche*
Die Faktoren, die in das Experiment einfließen sollen, müssen ausgewählt werden. Weiterhin muss bestimmt werden, in welchem Wertebereich die Faktoren untersucht werden sollen und welche Level in diesen Bereichen für die Durchführung des Experiments ausgewählt werden.

E-3 *Auswahl der Responsevariable*
Bei der Auswahl der Responsevariable ist zu beachten, dass diese aussagekräftig ist, was den untersuchten Prozess angeht. Eine wichtige Rolle bei der Auswahl spielt der Messfehler. Dieser entscheidet, wie groß die minimale Änderung der Variablen sein muss, damit sie festgestellt werden kann.

Die Auswahl der Faktoren, Level und Wertebereiche kann auch nach der Auswahl der Responsevariable bzw. gleichzeitig stattfinden. Zusammengenommen werden diese ersten drei Schritte als *vorexperimentelle Planung* bezeichnet.

E-4 *Auswahl des experimentellen Designs*
In diesem Schritt wird die Stichprobengröße, also die Anzahl der Wiederholungen festgelegt, die Reihenfolge der Durchführung der Testläufe bestimmt und darüber entschieden, ob Techniken zur Erhöhung der Genauigkeit des Experiments eingesetzt werden sollen. In der Praxis wird an dieser Stelle eine Software eingesetzt, die nach Eingabe aller notwendigen Parameter ein oder mehrere experimentelle Designs vorschlägt.

E-5 *Durchführen des Experiments*
Während der Durchführung des Experimentes ist es wichtig sicherzustellen, dass alles gemäß der Planung durchgeführt wird. Ein Fehler in der Durchführung kann das ganze Experiment hinfällig werden lassen.

E-6 *Statistische Analyse der Daten*
Statistische Methoden sollten für die Auswertung der Daten eingesetzt werden, da gezogene Schlussfolgerungen dann objektiv gezogen werden können. Es bietet sich wiederum der Einsatz von entsprechenden Softwarepaketen an. Wurde das experimentelle Design korrekt aufgestellt und bei der Durchführung des Experiments befolgt, so ist die anschließende Auswertung in der Regel mit einfachen statistischen Hilfsmitteln möglich. Einfache graphische Methoden sind oft gut geeignet, um die erhaltenen Daten zu analysieren und interpretieren.

E-7 *Schlussfolgerung und Empfehlung*
Nachdem die Analyse der Daten erfolgt ist, sollten aus den Ergebnissen praktisch verwertbare Schlussfolgerungen gezogen werden und diese als Empfehlung präsentiert werden. Bei der Präsentation können wiederum graphische Hilfsmittel eingesetzt werden.

2.5.3 DACE

Santner, Williams und Notz haben sich mit dem „Design and Analysis of Computer Experiments" (DACE) auseinander gesetzt [33]. Unter einem Computerexperiment verstehen sie dabei eine Simulation eines physikalischen Experimentes anhand einer

mathematischen Theorie. Statt des physikalischen Experiments wird dann das Computerexperiment durchgeführt. Gründe können beispielsweise ethische Bedenken sein, z.B. bei Experimenten mit radioaktiven Stoffen. Der gravierende Unterschied zwischen physikalischen Experimenten und Computerexperimenten ist dabei, dass Letztere deterministisch sind und somit Mechanismen zur Minimierung von Fehlern, wie die im DOE-Ansatz verwendete Randomisierung, Blocking und Wiederholung, hier sinnlos sind. Auch dort, wo Computerexperimente an Stelle von physikalischen Experimenten verwendet werden können, kann die Durchführung der Simulation mit erheblichem Aufwand verbunden sein. Ein Beispiel für einen rechenintensiven Simulationsprozess ist die Finite Elemente Analyse, die u.a. für die Simulation von Wetterdaten eingesetzt werden kann. Das Ziel von DACE ist es, mit so wenig Simulationsläufen wie möglich das Experiment erfolgreich durchzuführen. Dazu wird auf Interpolationstechniken zurückgegriffen, so dass die Ergebnisse möglichst vieler Simulationsläufe ausreichend genau vorhergesagt werden können, ohne sie tatsächlich berechnen zu müssen. Auch für den Einsatz von DACE gibt es geeignete Hilfsmittel in Form von entsprechenden Softwarepaketen. Ein solches Paket gibt es beispielsweise von Lophaven, Nielsen und Søndergaard [25, 24]. Dieses verwendet die sogenannte *Kriging*-Methode, um Ergebnisse von Simulationsläufen vorherzusagen. Dabei handelt es sich um einen Algorithmus, der ein stochastisches Prozessmodell verwendet. Ein Vorteil der Kriging-Methode ist, dass der *mittlere quadratische Fehler* (MSE) der Vorhersage angegeben werden kann.

2.5.4 Sequenzielle Parameter Optimierung (SPO)

Die *Sequenzielle Parameter Optimierung* (SPO) ist eine von Bartz-Beielstein vorgeschlagene Vorgehensweise, die Methoden aus DOE und DACE geschickt miteinander kombiniert, um auch nicht-deterministische Computerexperimente effizient durchführen zu können, wie insbesondere die Parameteroptimierung bei den randomisierten evolutionären Algorithmen [4]. SPO ist eine Vorgehensweise in 12 Schritten:

(S-1) *Vorexperimentelle Planung*
Das Ziel der Untersuchung wird festgelegt und die Sammlung der Daten wird geplant. Denkbare Ziele sind beispielsweise das Finden neuer Ergebnisse oder das Bestätigen bereits vorhandener.

(S-2) *Wissenschaftlichen Behauptung*
Das Ziel der Untersuchung wird in Form einer wissenschaftlichen Behauptung formuliert. Ein solche Behauptung wäre beispielsweise „Es existiert eine Parameterkonfiguration für einen Algorithmus, die zu besseren Ergebnissen führt als die Standardeinstellung".

(S-3) *Statistische Hypothese*
Es wird eine Hypothese aufgestellt, die durch das Experiment bestätigt oder verworfen werden soll. Auf das o.g. Beispiel angewendet könnte eine geeignete Hypothese beispielsweise lauten: „Parametereinstellung p_o führt bei Anwendung von Algorithmus A auf Problem P zu signifikant besseren Ergebnissen als Parametereinstellung p_{std}.

(S-4) *Spezifikation*

1. des Optimierungsproblems
2. der Nebenbedingungen
3. der Initialisierungsmethode des EA
4. der Abbruchmethode
5. des Algorithmus (wichtige Faktoren)
6. des initialen experimentellen Designs
7. des Berwertungsmaßes für die Performance

(S-5) *Experimentieren*
Die Versuche werden gemäß des experimentellen Designs durchgeführt und die erhaltenen Daten in geeigneter Weise gespeichert. Es ist dabei zu beachten, dass die Experimente wiederholt durchgeführt werden müssen, da die Auswertung mit DACE-Methoden von deterministischen Algorithmen ausgeht, hier aber randomisierte Verfahren vorliegen. In späteren Iterationen muss die Anzahl der Wiederholungen in geeigneter Weise erhöht werden, um bei der Auswertung eine faire Gewichtung der neu hinzukommenden Daten zu gewährleisten.

(S-6) *Statistische Modellierung und Vorhersagen*
In diesem Schritt wird ein stochastisches Prozessmodell aufgestellt und an die im Experiment gewonnenen Datenpunkte angepasst.

(S-7) *Auswertung und Visualisierung*
Das im letzten Schritt erstellte Modell wird nun ausgewertet. Die Ergebnisse können mit geeigneten Visualisierungsmethoden dargestellt werden. Es können beispielsweise Histogramme und Scatterplots verwendet werden, um Ausreißer festzustellen. Weiterhin können die Effekte und Interaktionen der Faktoren graphisch dargestellt werden. Bartz-Beielstein weist explizit darauf hin, dass mit den hier verwendeten Modellen nicht bewiesen werden kann, dass ein spezieller Faktor eine bestimmte Wirkung hat, es können lediglich Hinweise geliefert werden, wie in weiteren Experimenten zu verfahren ist.

(S-8) *Optimierung*
Fall sich die Ergebnisse des Experiments als geeignet herausgestellt haben, können damit neue Designpunkte in vielversprechenden Regionen des Suchraums gefunden werden, die in folgenden Experimenten dazu dienen, die bisher gewonnenen Erkenntnisse zu vertiefen und zu die Ergebnisse zu verbessern.

(S-9) *Abbruch?*
Falls die erhaltene Lösung gut genug ist oder die maximale Anzahl Iterationen erreicht ist, gehe zu Schritt (S-11).

(S-10) *Anpassen des Designs*
Das experimentelle Design wird anhand der in (S-8) beschriebenen Designpunkte angepasst und in einem weiteren Experiment durchgeführt. Es wird also mit Schritt (S-5) weitergemacht.

(S-11) *Verwerfen oder Akzeptieren der statistischen Hypothese*
Die Abschließende statistische Auswertung wird vorgenommen und die Ergebnisse zusammengefasst. Die statistische Hypothese aus Schritt (S-2) wird akzeptiert bzw. verworfen. Es sollte ein Vorher-Nachher-Vergleich der Parameterkonfigurationen durchgeführt werden.

(S-12) *Objektive Interpretation der Ergebnisse*
Abschließend muss festgestellt werden, ob das Ergebnis wissenschaftliche Bedeutung hat oder nicht. Dazu sollte eine objektive Interpretation der Evaluierung der statistischen Hypothese erfolgen.

Auch für SPO gibt es ein Softwarepaket von Bartz-Beielstein, welches den größten Teil des Designs und der Analyse automatisiert. Dieses Softwarepaket wird in Abschnitt 4.3 eingesetzt, um die Parameter eines in dieser Arbeit vorgestellten evolutionären Algorithmus geeignet einzustellen. Ein Beispiel für die Anwendung von SPO findet man bei Bartz-Beielstein [3], wo die Parameter eines Particle-Swarm-Optimization-Verfahrens optimiert werden.

Kapitel 3

Evolutionäre Kreuzungsminimierung

Dieses Kapitel erarbeitet einige evolutionäre Ansätze, um den Planarisierungsschritt des TSM-Modells zu lösen. Das Ziel ist dabei stets die Minimierung der erzeugten Dummy-Knoten und damit die Minimierung der daraus resultierenden Kreuzungen.

3.1 Evolutionäre Berechnung Maximaler Subgraphen

Wie bereits im Abschnitt 2.3.3 beschrieben wurde, wird die Planarisierung beim TSM-Modell in der Regel in zwei Schritten durchgeführt. Im ersten Schritt werden solange Kanten aus dem Graphen entfernt, bis er planar ist und im zweiten Schritt werden diese Kanten sukzessive wieder eingefügt, wobei die Anzahl der Kreuzungen minimiert werden sollte. Die Graphen, die durch das Entfernen von Kanten entstehen, werden *Subgraphen* genannt.

Definition 3.1. *Sei $G = (V, E)$ ein Graph. Ein* Subgraph *$G^* = (V^*, E^*)$ von G ist ein ist Teilgraph von G, für den $V^* = V$ gilt.*

Der Graph, für den der Subgraph gebildet wird, wird dessen *Ursprungsgraph* genannt. Die Knotenmenge eines Subgraphen ist die gleiche, wie die des Ursprungsgraphs. Die Kantenmenge ist hingegen eine Teilmenge der Kanten des Ursprungsgraphs.

In diesem Abschnitt wird zunächst ein evolutionärer Algorithmus, genauer ein genetischer Algorithmus beschrieben, der nach einen maximalen planaren Subgraphen sucht. Darauf aufbauend werden anschließend einige GAs ausgearbeitet, die den gesamten Planarisierungsschritt evolutionär lösen.

Um einen GA zu entwerfen, der die geforderte Aufgabe lösen kann, sind folgende Vorüberlegungen notwendig:

1. Was ist eine geeignete Repräsentation für Subgraphen als Individuen eines GAs?

2. Wie können die Subgraphen bewertet werden, was ist also eine geeignete Fitnessfunktion für die Bewertung eines Subgraphs?

Um Subgraphen darzustellen, könnte ein Bitvektor benutzt werden, in dem es für jede Kante des Ursprungsgraphens ein Bit gibt, das anzeigt, ob die Kante in den Subgraphen übernommen wird. Der Vorteil dieser Darstellung ist, dass sie einfach gehalten ist, wenig Speicher benötigt und die Verwendung einfacher Operatoren ermöglicht, wie beispielsweise den in Abschnitt 2.2 vorgestellten One-Point-Crossover.

Eine alternative Repräsentation dazu ist die durch Permutationen. Diese können ausgehend von einem Subgraphen mit leerer Kantenmenge die Reihenfolge angeben, in der die Kanten eingefügt werden, sofern dies die Planarität nicht verletzt. Dies hat allerdings den Nachteil, dass nicht die einfachen Operatoren benutzt werden können, die bei einer Bitvektor-Repräsentation möglich sind. Ein Crossover für einen Bitvektor angewandt auf Permutations-Individuen erzeugt im Allgemeinen nicht wieder Permutationen. Es müssen daher die komplizierteren Operatoren eingesetzt werden, wie sie in Abschnitt 2.2 beschrieben sind.

Wäre das Ziel lediglich die Berechnung maximaler Subgraphen, fiele die Wahl sicherlich auf eine Bitvektor-Darstellung. Da diese aber nur den ersten Schritt darstellt und im weiteren die Einfügereihenfolge optimiert werden soll, ist die Repräsentation durch Permutation vorzuziehen, da Permutationen auf natürliche Art Reihenfolgen darstellen. Abbildung 3.1 zeigt ein Beispiel für die Bildung von maximalen Subgraphen mit Hilfe von Permutationen. In Bild 1 ist der Graph dargestellt, für den der maximale planare Subgraph berechnet werden soll. Um den Zusammenhang zwischen Permutationen und Kanten eindeutig herzustellen, müssen die Kanten nummeriert werden. Dies ist in Bild 2 zu sehen. Die letzten beiden Bilder zeigen die resultierenden maximalen Subgraphen für zwei verschiedene Permutationen. Offensichtlich können die Kanten mit den Nummern 6 und 8 nicht am Schluss eingefügt werden, ohne dass der Subgraph seine Planarität verliert.

Ein GA für Permutationen

An dieser Stelle wird zunächst ein GA beschrieben, der Permutationen als Individuen verwendet. Dieser wird im Folgenden als „Framework" für die verschiedenen Anwendungen verwendet. Es wird im wesentlichen nur noch die Fitnessfunktion angepasst. Dies entspricht der Philosophie der *Blackboxoptimierung*, also der Optimierung ohne problemspezifisches Wissen einzusetzen, die als erster Ansatz bei Anwendung evolutionärer Verfahren oft schon gute Ergebnisse liefert. Sollten die Ergebnisse nicht zufriedenstellend sein, kann das Verfahren anschließend durch das Einbringen problemspezifischer Lösungsstrategien verbessert werden.

Für den ersten Ansatz einer evolutionären Kreuzungsminimierung wird hier entsprechend ein möglichst einfacher GA verwendet. Dieser lässt sich wie folgt beschreiben:

- Als Individuen werden Permutationen verwendet. Wie bereits zuvor erwähnt ist der Grund hierfür, dass die Individuen letztendlich die Reihenfolgen für das Einfügen von Kanten repräsentieren werden. Die Kanten des ursprünglichen Graphen werden durchnummeriert. Die Elemente der Permutationen können dann ganze Zahlen sein, die jeweils die Kantennummer darstellen.

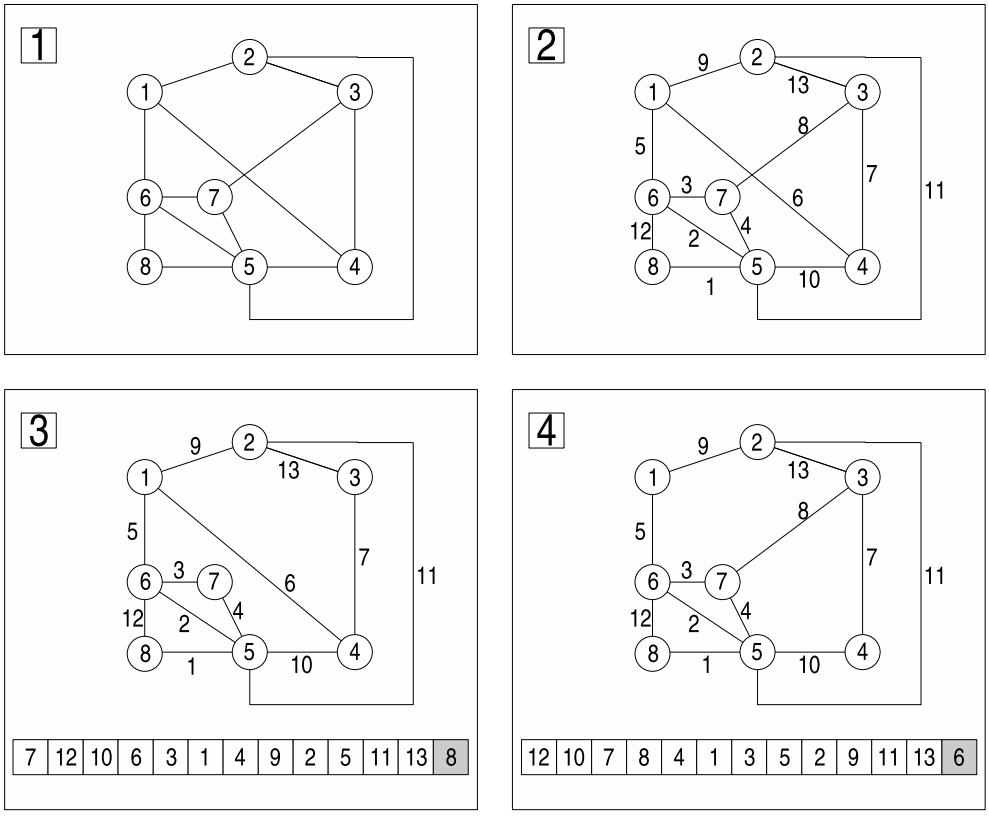


Abbildung 3.1: Beispiel für zwei verschiedene Einfügereihenfolgen. 1: Der Ursprungsgraph. 2: Die Kantennummerierung. 3: Maximaler planarer Subgraph zu Permutation $(7, 12, 10, 6, 3, 1, 4, 9, 2, 5, 11, 8)$. 4: Maximaler planarer Subgraph zu Permutation $(12, 10, 7, 8, 4, 1, 3, 5, 2, 9, 11, 6)$.

- Die Population dieses GAs wird auf konstanter Größe gehalten. Die Größe der Population ist einer der Laufzeitparameter, die von vor Beginn der Optimierung eingestellt werden können.
- Der GA sucht nach dem Minimum. Das bedeutet, das Individuen, die einen kleineren Fitnesswert haben „besser" sind als solche, die einen großen Fitnesswert besitzen.
- Bei der Initialisierung werden folgenden Aufgaben erfüllt:
  - Die Kanten des ursprünglichen Graphen erhalten eine Reihenfolge zugewiesen und werden durchnummeriert (s.o.).
  - Die Population wird initialisiert. Dazu werden entsprechend der Populationsgröße eine Menge von Individuen erzeugt, die *Anfangspopulation*. Jedes dieser Individuen ist eine Permutation der Zahlen $1, \ldots, n$, wobei n die Anzahl der Kanten im ursprünglichen Graphen ist. Die Individuen werden zufällig erzeugt. Dazu werden in einer anfangs geordneten Folge von n Zahlen wiederholt zwei zufällig ausgewählte Elemente miteinander vertauscht.

  - Die Anfangspopulation wird gemäß der Fitnessfunktion *evaluiert*. Das heißt, es wird jedem Individuum der Anfangspopulation sein Fitnesswert zugewiesen.

- Nach der Initialisierung beginnt die eigentliche Iteration über die Generationen der Population. In jeder Generation wird zunächst gemäß dem Vorgehen bei GAs die Abbruchbedingung überprüft. Dieser GA verwendet als Abbruchbedingung die Anzahl der Fitnessauswertungen, die insgesamt seit Start der Optimierung ausgeführt wurden, also inklusive der Evaluierung Anfangspopulation. In jeder Generation werden die folgenden Schritte durchgeführt:

  1. Es werden die Eltern der Nachfolgegeneration in den Mating-Pool ausgewählt. Die Auswahl erfolgt dabei fitnessproportional mit dem Roulettewheel-Verfahren. Das bedeutet, dass die Wahrscheinlichkeit eines Individuums, als Elternteil für die Nachfolgegeneration ausgewählt zu werden, umgekehrt proportional zu seinem Fitnesswert ist, da der Fitnesswert minimiert werden soll.
  2. Die Individuen, die in den Mating-Pool Einzug gefunden haben, werden nun paarweise miteinander rekombiniert. Dazu wird hier der Crossover-Operator von Davis verwendet, der in Abschnitt 2.2 beschrieben ist (s. auch Abbildung 2.5). Die Crossover-Wahrscheinlichkeit, also diese Wahrscheinlichkeit, mit der überhaupt ein Crossover stattfindet, ist ein weiterer Laufzeitparameter.
  3. Nach der Rekombination werden die generierten Nachkommen mit einer gewissen Wahrscheinlichkeit, der Permutationswahrscheinlichkeit, noch permutiert. Die Permutation wird dabei realisiert durch das Vertauschen zweier zufällig ausgewählter Elemente der entsprechenden Permutation. Auch die Permutationswahrscheinlichkeit ist ein Laufzeitparameter, der in der Konfiguration des GAs eingestellt werden kann.
  4. Anschließend werden alle neu erzeugten Individuen evaluiert, sie bekommen als ebenfalls ihren Fitnesswert zugewiesen.

- Ist die maximale Anzahl von Fitnessauswertungen erreicht, so wird das beste Individuum, das im Laufe der Optimierung Teil der Population war, verwendet, um die Ausgabe zu erzeugen. Das beste Individuum ist in diesem Fall dasjenige, mit dem kleinsten Fitnesswert. Es sei an dieser Stelle darauf hingewiesen, dass sich das absolut beste Individuum nicht zwangsläufig auch zum Schluss der Optimierung in der Population befinden muss. Es kann sein, dass es durch Permutation wieder aus der Population verschwunden ist.

Die folgende Tabelle fasst nochmal die Laufzeitparameter des GAs zusammen:

Parameter	Typ	Typischer Wert
Populationsgröße	Integer	100
Poolgröße	Integer	20
Crossover-Wahrscheinlichkeit	Fließkomma	0,8
Mutations-Wahrscheinlichkeit	Fließkomma	0,01

Fitnessfunktion für maximale Subgraphen

Mit dem im letzten Abschnitt beschriebenen Grundgerüst für den in dieser Arbeit verwendeten GA ist es nun nicht mehr schwer, ein Verfahren für die Suche nach dem maximalen Subgraphen zu entwickeln. Es muss lediglich eine geeignete Fitnessfunktion gefunden werden und zum Schluss die richtig Ausgabe erzeugt werden.

Die Fitnessfunktion muss so beschaffen sein, dass Subgraphen umso kleinere Fitnesswerte bekommen, je mehr Kanten sie haben, da der GA die Fitnesswerte zu minimieren versucht. Dazu wird zunächst der Subgraph des ursprünglichen Graphen erzeugt, der keine Kante beinhaltet. In diesen werden nun die Kanten gemäß der Reihenfolge der Permutation wieder eingefügt. Dabei wird nach jedem Kanteneinfügen überprüft, ob der Graph noch planar ist. Der Test auf Planarität ist effizient möglich [18]. Solche Kanten, die zu einem nichtplanaren Subgraphen führen, werden ausgelassen. Der Fitnesswert berechnet sich nun aus der Differenz aus der Anzahl von Kanten im ursprünglichen Graphen und der Anzahl von Kanten im soeben gebildeten Subgraphen:

$$\begin{aligned} \text{Fitness} &= \text{Anzahl Kanten im Ursprungsgraph} - \text{Anzahl Kanten im Subgraph} \\ &= |E| - |E^*| \end{aligned}$$

Dieser Fitnesswert hat die Eigenschaft, dass er immer größer als Null ist, da der ursprüngliche Graph nicht planar ist. Weiterhin haben Subgraphen mit mehr Kanten kleinere Fitnesswerte. Somit ist der berechnete Wert für den oben beschriebenen GA tauglich.

Für die Rückgabe des GAs bilden wir aus dem besten Individuum genau wie in der Fitnessauswertung den Subgraphen aus der Permutation. Es kann dann entweder der Subgraph selbst zurückgegeben werden oder lediglich die Anzahl von Kanten darin, je nachdem, was gefragt ist.

3.2 Evolutionäre Planarisierung

Der letzten Abschnitt hat beschrieben, wie ein GA für die Berechnung eines maximalen Subgraphen erstellt werden kann. Es wurde ein Framework mit Permutation als Individuen vorgestellt. Auf diesem aufbauend beschreibt dieser Abschnitt zwei Varianten von GAs, die den Planarisierungsschritt durchführen und dabei die Anzahl der Kreuzungen minimieren.

Im Gegensatz zu dem zuvor beschriebenen GA für die Suche nach einem maximalen Subgraphen muss ein Individuen nun nicht mehr nur einen Subgraphen repräsentieren, sondern eine komplette Planarisierung des Ursprungsgraphen.

Fitnessfunktion

Die Fitnessfunktion soll wiederum für „gute" Individuen kleinere Fitnesswerte ausgeben. Ein Individuum ist nun dann als „besser" anzusehen als ein anderes, wenn es zu einer Planarisierung des Graphen führt, die weniger Kreuzungen beinhaltet als die des anderen Individuums. Planarisierungen mit gleicher Kreuzungszahl sollen hier auch als gleich gut beurteilt werden.

Es ist naheliegend und sinnvoll, als Fitnesswert die Anzahl der Kreuzungen zurückzugeben, die man erhält, wenn die zu der Permutation gehörende Planarisierung durchgeführt wird. Dazu muss diese Planarisierung durchgeführt werden und anschließend die entstandenen Kreuzungen gezählt werden.

Ein Problem gibt es jedoch, wenn die Anzahl der Kreuzungen als Fitnessfunktion verwendet wird. Es kann dann viele Individuen mit demselben Fitnesswert geben. Der Fitnesswert stellt dann kein Kriterium dar, nach dem die Population eindeutig geordnet werden kann. Dies ist aber für die Betrachtung von Evolutionären Algorithmen im Allgemeinen erwünscht. Als Lösung dieses Problems werden solche Individuen, die denselben Fitnesswert bekommen, lexikographische geordnet.

Der Algorithmus für die Durchführung einer Planarisierung bekommt als Eingabe den Graphen G, der Planarisiert werden soll, zusammen mit einer Nummerierung seiner Kanten und einer Permutation. Die Ausgabe besteht aus dem gemäß der Permutation planarisierten Graphen. Die Planarisierung wird wie folgt durchgeführt:

1. Es wird der Subgraph von G gebildet, der keine Kante enthält.
2. Nun werden die Kanten in der Reihenfolge der Permutation durchlaufen. Für jede Kante wird wie folgt verfahren:
   (a) Füge die Kante in den Subgraphen ein.
   (b) Teste, ob der Graph planar ist. Es könne zwei Fälle auftreten:
      i. Der Subgraph ist planar. Es passiert nichts.
      ii. Der Subgraph ist nicht planar. Die Kante wird wieder entfernt und in einer geordneten Liste gespeichert.
3. Es liegt nun ein planarer Subgraph gemäß der Permutation des Individuums vor. Füge die noch fehlenden Kanten aus der Liste in der gegebenen Reihenfolge in den Subgraphen ein. Ersetze dabei nach dem Einfügen jeder Kante die entstandenen Kreuzungen durch Dummy-Knoten.
4. Die Ausgabe ist der so gebildete planarisierte Graph.

Der Algorithmus lässt offen, nach welcher Methode die Kanten eingefügt werden sollen. Es wurden in Abschnitt 2.3.2 bereits zwei Methoden zum Einfügen von Kanten eingeführt. Diese sollen hier eingesetzt werden. Ein Vorteil dabei ist, dass diese bereits die Eigenschaft haben, entstandene Kreuzungen durch Dummy-Knoten zu ersetzen, so dass keine zusätzlichen Maßnahmen hierzu durchgeführt werden müssen.

Je nachdem, welche der beiden Methoden eingesetzt werden, unterscheiden wir zwei Varianten des GAs:

FIX Es wird der Algorithmus zum Einfügen von Kanten bei fixer kombinatorischer Einbettung verwendet.

VAR Es wird der Algorithmus zum Einfügen von Kanten bei variabler kombinatorischer Einbettung verwendet.

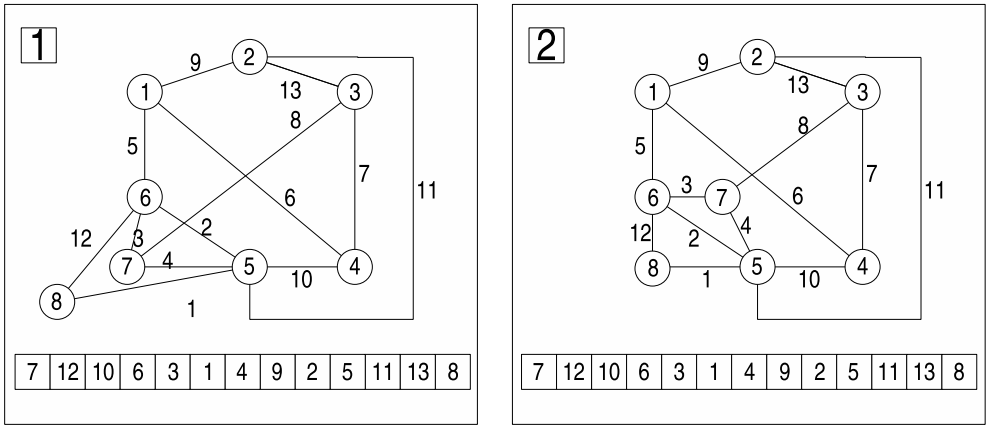


Abbildung 3.2: Mögliche unterschiedliche Ergebnisse der Varianten FIX und VAR. 1: Ungünstiges Ergebnis für FIX. 2: Besseres Ergebnis für VAR.

Dass die beiden Varianten tatsächlich zu unterschiedlichen Berechnungen kommen können, ist in Abbildung 3.2 dargestellt. Bild 1 zeigt ein ungünstiges Ergebnis der Variante FIX. Wurde zuvor die kombinatorische Einbettung so festgelegt, dass Knoten 7 auf die „ungünstige Seite" der Kante 2 angeordnet ist. Dies so führt später beim Einfügen der Kante 8 zu einer unnötigen Kreuzung, da die kombinatorische Einbettung nicht mehr geändert wird. Die Variante VAR hingegen wählt unter allen kombinatorischen Einbettungen die günstigste aus, so dass es zu einem besseren Ergebnis kommt, wie es in Bild 2 zu sehen ist.

Berechnung der Ausgabe

Wurde der Optimierungslauf erfolgreich durchgeführt, so kennt man das Individuum, also die Permutation, die zum besten Ergebnis geführt hat. Da der GA den planarisierten Graphen zurückgeben soll, muss entweder der beste planarisierte Graph gespeichert werden, der für die Fitnessauswertung erzeugt wurde, oder man berechnet ihn neu. Das Verfahren ist in diesem Fall dasselbe.

3.3 Kantengewichte

Dieser Abschnitt stellt zwei weitere Varianten von GAs vor, die den Planarisierungsschritt durchführen. Der Unterschied zu den bisher dargestellten GAs besteht darin, dass in den neuen Varianten künstlich Kantengewichte eingeführt werden, um das Einfügen der Kanten zu beeinflussen. Beide Einfügealgorithmen beruhen auf der Suche nach kürzesten Wegen. Die hier vorgestellten Varianten werden dies erweitern auf die Suche nach billigsten Wegen in der Hoffnung, dass dadurch gute Lösungen gefunden werden können, die ansonsten unentdeckt geblieben wären.

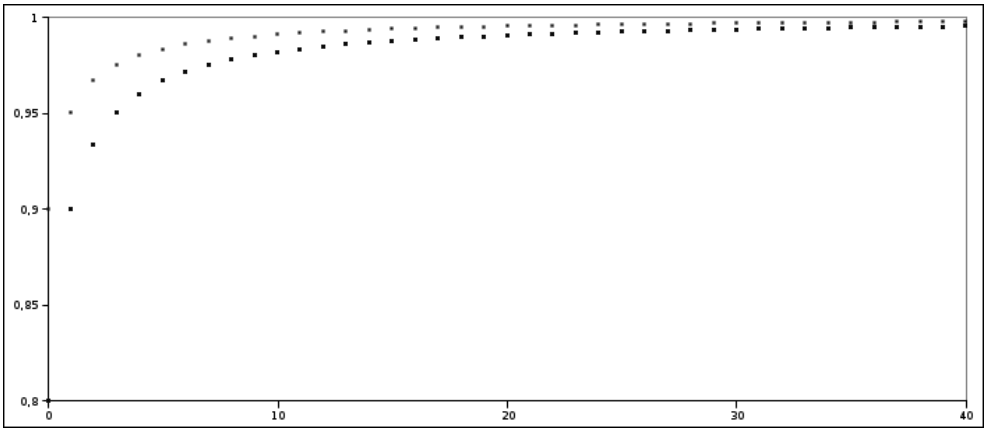


Abbildung 3.3: Die Kantengewichte, wie sie für $\epsilon = 0.1$ (oben) bzw. $\epsilon = 0.2$ (unten) generiert werden.

Kantengewichte

Dieser Abschnitt beschreibt, wie die Zuordnung von Kantengwichten realisiert wird. Da die Graphen, die planarisiert werden sollen, im Allgemeinen keine geeigneten Kantengewichte besitzen, müssen diese künstlich berechnet und den Kanten zugeordnet werden.

Dazu muss zunächst für jede Kante des Ursprungsgraphens ein Kantengewicht erzeugt werden. Diese sollten derart gestaltet sein, dass im weiteren Verlauf keine zwei verschiedenen Kanten dieselben Kosten verursachen, damit in jedem Fall ein eindeutiger kostengünstigster Pfad gefunden werden kann. Um n Kantengewichte $w_1, \ldots, w_n$ mit den geforderten Eigenschaften zu generieren, kann die Formel

$$w_i = 1 - \frac{1}{i \times \epsilon}$$

verwendet werden. Darin ist ϵ ein Skalierungsfaktor, der angibt, in welchem Bereich um 1 herum die Gewichte generiert werden sollen. In Abbildung 3.3 sind die erzeugten Gewichte für $\epsilon = 0,1$ und $\epsilon = 0,2$ dargestellt. Wie leicht zu erkennen ist, sind die meisten Gewichte „in der Nähe" von 1 angesiedelt. Dies hat zur Folge, dass ein großer Teil der Kanten so behandelt wird, als gäbe es keine Kantengewichte. Der Vorteil bei diesen Kanten ist, dass der Algorithmus eindeutig eine Kante nach dem etwas geringeren Gewicht auswählt, wenn mehrere zur Auswahl stehen. Es kann also durch die Zuweisung der Kantengewichte manipuliert werden, welcher Weg bei gleicher Kantenlänge den Vorrang haben soll. In den GAs ohne Kantegewichte gibt es eine solche Einflussmöglichkeit nicht. Die wenigen Kanten, die ein sehr geringes Kantengewicht zugewiesen bekommen, werden bei der Suche nach günstigen Wegen zum Kanteneinfügen bevorzugt behandelt. Wichtig ist nun, eine gute Zuordnung für die Kantengewichte zu finden, damit die Suche zu guten Ergebnissen kommen kann.

Günstigste Wege

Bisher ist noch gar nicht klar, wie die Kantengewichte überhaupt das Einfügen einer Kante beeinflussen. Die Kantengewichte sind zunächst lediglich auf dem Ursprungsgraphen definiert. Von dort können sie problemlos auf die in der Fitnessauswertung gebildeten Subgraphen abgebildet werden. Das Einfügen einer Kante geschieht aber bei der Variante FIX über den erweiterten dualen Graphen. Auf diesen haben die Kantengewichte bisher noch keinerlei Auswirkung. Es können jedoch die Kantengewichte des Ursprungsgraphen leicht auf den erweiterten dualen Graphen abgebildet werden. Die Gewichte lassen sich wie folgt bei Bedarf leicht ermitteln:

- Jede Kante des dualen Graphen hat als Gewicht das Kantengewicht der Kante im Ursprungsgraphen, die zwischen den beiden Faces liegt, welche durch die Kante des dualen Graphen verbunden werden.
- Die erweiterten Kanten zum Start- und zum Endknoten erhalten jeweils Gewicht 0. Dies ist sinnvoll, denn diese Kanten erzeugen beim Einfügen der Kante in den Subgraphen keine Kreuzung und keinen Dummy-Knoten.

Nun kann im erweiterten dualen Graphen mit einem der bekannten effizienten Algorithmen zur Suche von kantengünstigsten Wegen gearbeitet werden.

Die Variante VAR kann analog zu der beschriebenen Methode ebenfalls um Kantengewichte erweitert werden.

GAs mit Kantengewichten

Es wurde bisher erläutert, wie Kantengewichte generiert werden können und wie sie eingesetzt werden, um Kanten in Subgraphen einzufügen. Nun muss noch geklärt werden, wie die GAs aufzubauen sind, um die beschriebenen Verfahren einzusetzen.

Zunächst einmal werden die Kantengewichte nur beim Einfügen von Kanten in die Subgraphen verwendet, also im zweiten Schritt der Planarisierung. Dieser findet in den GAs nur während der Fitnessauswertung und bei der Berechnung des Endergebnisses statt. Da eine optimale Zuordnung der Gewichte zu den Kanten nicht bekannt ist, bietet es sich an, diese durch den GA optimieren zu lassen. Dazu wird in den Individuen neben der Permutation, welche die Kantenreihenfolge repräsentiert, eine weitere Permutation mit den Kantengewichten hinterlegt. Entsprechend sind die Operatoren des GAs wie folgt anzupassen:

Evaluation Die Berechnung der Fitnesswerte wird nur dahingehend verändert, dass nun die Wege im erweiterten dualen Graphen mit einer anderen Methode gesucht werden. Die Anzahl der erzeugten Kreuzungen ist weiterhin das Kriterium für den Fitnesswert.

Selektion Die Selektion muss nicht verändert werden, da hier lediglich Individuen ausgewählt werden. Es wird nicht mit Ihnen gearbeitet.

Rekombination Der Crossover-Operator muss nun zwei Permutationen kreuzen. Dies wird dadurch realisiert, dass die beiden Permutationen, die in einem Individuum gespeichert sind, getrennt und unabhängig voneinander gekreuzt werden.

Dies ist sinnvoll, da kein Zusammenhang zwischen der Kantenreihenfolge und der Kantengewichte bekannt ist. Es wird weiterhin der Crossover-Operator von Davis für beide Operationen verwendet. Die Crossover-Wahrscheinlichkeit wird für beide Permutationen gleich angenommen.

Mutation Der Mutations-Operator kann ebenfalls für beide Permutationen unabhängig voneinander durchgeführt werden. Die Mutationswahrscheinlichkeit wird für beide Permutationen gleich angenommen.

Elemination Die fitnessproportionale Elemination muss nicht angepasst werden, da lediglich die Fitnesswerte der Individuen für die Auswahl verwendet werden.

Diese GAs verwenden alle bereits bekannten Laufzeitparameter. Zusätzlich kann der Parameter ϵ für die Generierung der Kantengewichte zu den Laufzeitparametern gezählt werden. Wir werden im Folgenden jedoch stets den Wert $\epsilon = 0,1$ annehmen.

Mit dem so beschriebenen GA können wiederum zwei Variante implementiert werden:

W-FIX Es wird der erweiterte Algorithmus zum Einfügen von Kanten bei fixer kombinatorischer Einbettung unter Zuhilfenahme von Kantengewichten verwendet.

W-VAR Es wird der erweiterte Algorithmus zum Einfügen von Kanten bei variabler kombinatorischer Einbettung unter Zuhilfenahme von Kantengewichten verwendet.

Kapitel 4

Experimente

Dieses Kapitel beschreibt die Experimente, die mit den genetischen Algorithmen zur Kreuzungsminimierung durchgeführt wurden. Mit den Experimenten wurden mehrere Ziele verfolgt. Zunächst sollte eine möglichst gute Einstellung für die Laufzeitparameter gefunden werden. Die dazu verwendete Vorgehensweise ist in Abschnitt 4.3 beschrieben. Mit den gefundenen guten Parametereinstellungen sollten anschließend die vier implementierten Varianten des genetischen Algorithmus miteinander verglichen werden. Abschnitt 4.4 beschreibt stellt die Ergebnisse dieses Experiments dar. Im letzten Abschnitt 4.5 werden die Resultate eines weiteren Experiments beschrieben, bei dem eine Variante des genetischen Algorithmus die Kreuzungsminimierung für eine große Menge von Graphen aus der Rome-Library durchführt. Diese Rome-Library wird zuvor in Abschnitt 4.1 beschrieben.

4.1 Die Rome-Library

Soll die Leistungsfähigkeit eines Algorithmus dargestellt werden, so kann dies auf zweierlei Wegen geschehen. Zum einen kann das Laufzeitverhalten des Algorithmus hergeleitet werden und zum anderen kann der Algorithmus in Experimenten ausprobiert werden. Da es sich bei den implementierten genetischen Algorithmen um randomisierte Verfahren handelt, kann das Laufzeitverhalten nur sehr allgemein unter der Annahme von u.U. unrealistischen Voraussetzungen, welche die Eingabe betreffen, hergeleitet werden. Die Alternative dazu, das Experiment, bietet sich als Ausweg an. Damit in einem Experiment aussagekräftige Ergebnisse erzielt werden können, die auch mit anderen Algorithmen vergleichbar sind, ist es notwendig, ein Menge von *Benchmark-Daten* zu haben, die als standardisierte Eingabe für alle in Frage kommenden Algorithmen verwendet werden kann. Im Bereich des Graphenzeichnens hat sich die sogenannte *Rome-Library* als so eine Menge von Eingabedaten, in diesem Fall Graphen, als Quasi-Standard durchgesetzt. Es bereits zahlreiche Ergebnisse, die sich auf diese Menge von Graphen beziehen. Gutwenger und Mutzel haben eine Kreuzungsminimierung der Graphen der Rome-Library mit den einigen aktuellen Verfahren bereits durchgeführt [15]. Somit liegen bereits Vergleichswerte vor, gegen die das Ergebnis eines Experiments mit unserem genetischen Algorithmus verglichen

werden kann. Die Rome-Library wurde erstmals 1997 von Di Batista et al. verwendet um vier Algorithmen zum Zeichnen von Graphen miteinander zu vergleichen [5]. Sie besteht aus 11582 Graphen, die alle zwischen 10 und 100 Knoten besitzen. Diese Graphen wurden aus einer Grundmenge von 112 Graphen generiert. Das Ergebnis ist in den beiden Abbildungen 4.1 und 4.2 dargestellt.

Abbildung 4.1 zeigt die Verteilung der Knotenanzahlen unter den Graphen. Auf der X-Achse sind die Knotenanzahlen aufgetragen, die in der Menge der Graphen vorkommen. Darüber befindet sich jeweils ein Balken, dessen Höhe angibt, wie viele Graphen mit dieser Knotenzahl existieren.

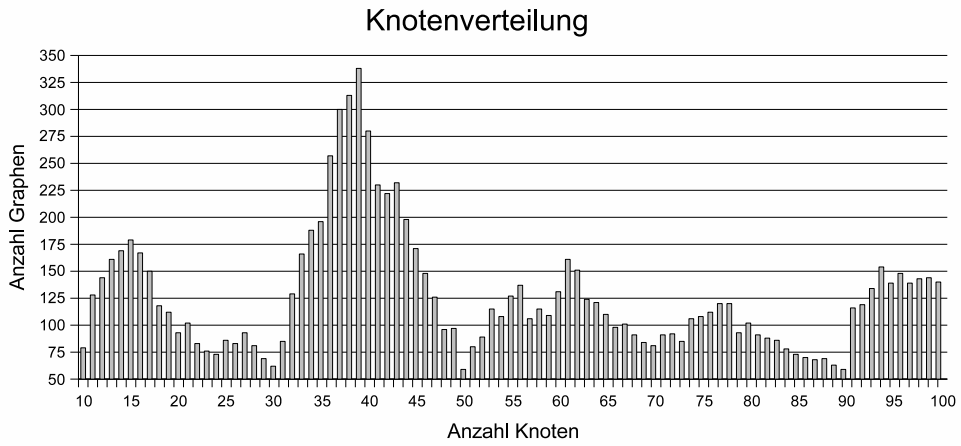


Abbildung 4.1: Verteilung der Knotenanzahlen in der Rome-Library.

Abbildung 4.2 zeigt hingegen die mittleren Kantenanzahlen an. Auf der X-Achse sind wiederum die vorkommenden Knotenanzahlen aufgeführt. Die Balken geben nun aber die mittlere Kantenanzahl aller Graphen der Rome-Library mit dieser Knotenanzahl an.

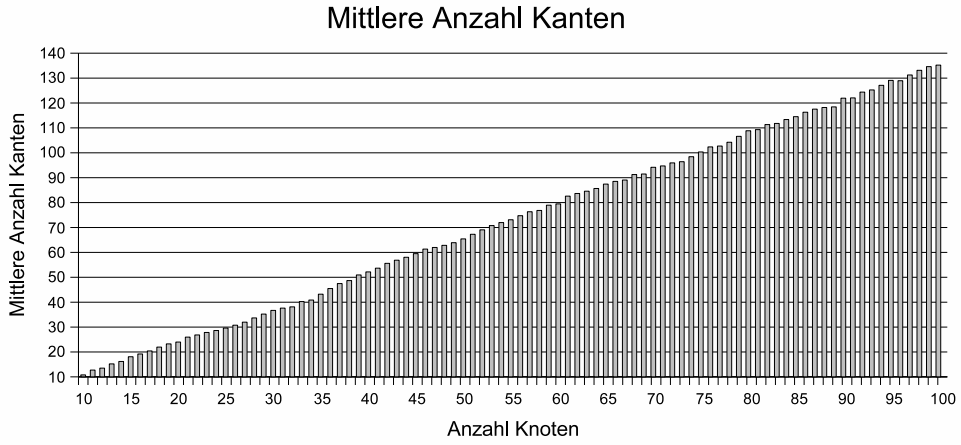


Abbildung 4.2: Verteilung der mittleren Kantenanzahlen in der Rome-Library.

Die Graphen dieser Grundmenge wurden so ausgewählt, dass sie solchen Graphen entsprechen, die alltäglich in Datenbank–Anwendungen oder in der Softwaretechnik auftreten. Von den Graphen der Rome-Library sind 8249 nicht planar. Dieses ist also die Untermenge, die für ein Experiment im Bereich Planarisierung von Interesses ist.

4.2 Langzeitexperiment

Bevor die eigentliche Parameteroptimierung der evolutionären Algorithmen in Angriff genommen wird, ist es sinnvoll, einige Testläufe mit Standardparametern durchzuführen. Oft werden dabei schon gute Ergebnisse erreicht. Sollten diese bereits gut genug sein, kann in manchen Fällen die Parameteroptimierung entfallen. Dies wird in der Praxis oft so gehandhabt, da methodische Vorgehensweisen zur Verbesserung der Konfiguration, wie beispielsweise SPO, noch nicht lange zur Verfügung stehen und somit vielen Anwendern noch fremd sind. Die Absicht der hier beschriebenen Untersuchungen ist jedoch nicht, eine Parameteroptimierung auszuschließen, sonder vielmehr sinnvolle Rahmenbedingungen für diese abzustecken.

Das Hauptziel der hier beschriebenen Testläufe ist es, die Abbruchbedingung für den GA zu bestimmen. Im konkreten Fall bedeutet das, die Anzahl der Fitnessauswertungen, nach denen der genetische Algorithmus abbrechen soll, zu bestimmen. Ist die Anzahl zu niedrig gewählt, kann der GA die Evolution der Individuen nicht weit genug vorantreiben, um gute Ergebnisse zu erhalten. Ist die Anzahl zu hoch gewählt, dauert die Optimierung für den praktischen Einsatz unter Umständen zu lange. Es gilt daher diese beiden Kriterien so gegeneinander abzuwägen, dass relativ gute Ergebnisse in „vernünftiger" Zeit erhalten werden. Was dabei „relativ gute Ergebnisse" und „vernünftige Zeit" sind, ist von der jeweiligen Anwendung abhängig. So ist es beispielsweise vorstellbar, dass einige Tage Rechenzeit auf den schnellsten verfügbaren Rechensystemen für einige industrielle Anwendungen lohnend und sinnvoll sind. Für den Privatanwender oder ein kleines Unternehmen wäre dies jedoch nicht finanzierbar.

Da es sich bei GAs um zufallsbasierte Verfahren handelt, kann natürlich keine gute Lösung durch die Entscheidung über ein Abbruchkriterium garantiert werden, im Mittel sollen aber gute Ergebnisse wahrscheinlich sein.

Um die Frage nach einer geeigneten Anzahl von Fitnessauswertungen beantworten zu können, sind sehr lange Testläufe geeignet. Es ist einzusehen, dass aus zu kurzen Testläufen keine Information für ein geeignetes späteres Abbruchkriterium gefunden werden kann, da keine gesicherte Information über den zukünftigen Verlauf der Optimierung daraus gewonnen werden kann.

Die Testläufe müssen also lang genug sein, um über den späteren Abbruchzeitpunkt hinaus zu laufen. Unglücklicherweise ist zu diesem Zeitpunkt ja noch unbekannt, wann dies sein wird, denn gerade das soll ja ermittelt werden. Als Notlösung wird der Testlauf deutlich länger laufen gelassen, als dies für die spätere Anwendung als vernünftig erachtet wird.

In dieser Arbeit wird der Langzeitlauf, wie auch später die Parameteroptimierung, exemplarisch nur für die Variante VAR durchgeführt. Diese wurde ausgewählt, da sie auf dem optimalen Kanteneinfügen über alle Einbettungen beruht, die ihre Leistungsfähigkeit bereits unter Beweis gestellt hat [15]. Es sind damit bessere Ergebnisse zu

erwarten als mit der Variante FIX. Dabei ist sie effizient genug, um für Langzeitläufe in Frage zu kommen. Aufgrund des komplizierteren Einfügens ist zu erwarten, das die gewichteten Varianten WFIX und WVAR langsamer sind. Außerdem ist noch nichts über die Güte ihrer Lösungen bekannt, so dass sie für den gewünschten Zweck nicht so gut geeignet sind.

Das Ziel aller Optimierungen dieser Arbeit sind die Graphen der in Abschnitt 4.1 vorgestellten Rome-Library. Es ist intuitiv klar, dass ein evolutionärer Ansatz mit den bekannten heuristischen Ansätzen nicht im Bezug auf die Geschwindigkeit konkurrieren kann. Um die Zeit abschätzen zu können, die der GA benötigt, wurden einige kurze Testläufe durchgeführt.

Alle Experimente wurden entweder auf einem Einzelplatzrechner (A1) durchgeführt oder auf der Rechenanlage des Lehrstuhls 11 des Fachbereichs Informatik der Universität Dortmund (A2). Im Folgenden werden die technischen Daten umrissen:

A1 Bei dem Einzelplatzrechner handelt es sich um einen handelsüblichen PC mit einem AMD Sempron 3000+ Prozessor und 1 GB Ram. Als Betriebssystem kam *Ubuntu Linux* in der Version *Breezy Badger x86_ 64* zum Einsatz.

A2 Das Rechensystem des Lehrstuhls 11 des Fachbereichs Informatik der Universität Dortmund ist ein Batchsystem, welches die freie Rechenzeit der dort verfügbaren Arbeitsplatzrechner nutzt. Bei den Rechnern handelt es sich ebenfalls um handelsübliche Rechner, deren technischen Daten variieren stark. Es sind Rechner mit 2,4 GHz Prozessor und 1 GB Ram vorhanden, aber auch solche, mit nur 1 GHz Taktfrequenz und 385 MB Ram. Das Betriebssystem ist einheitlich *Debian Linux* auf allen Rechnern. Die Obergrenze für die am Stück zur Verfügung stehende CPU-Rechenzeit ist auf 5 Stunden auf einem 1,2 GHz Rechner eingestellt. Für andere Prozessoren wird diese Zeit entsprechend angepasst.

Für die Langzeitexperimente wurde die Begrenzung der Rechenzeit temporär aufgehoben, so dass auch Läufe durchgeführt werden konnten, die wesentlich länger waren.

Für die Läufe wurden zufällig einig der nicht-planaren Graphen aus der Rome-Library ausgewählt.

Um die Langzeitläufe nicht mit völlig ungeeigneten Parametern durchzuführen, wurden zuvor einige Parametereinstellungen in kurzen Experimenten erprobt. Durch diese kurzen Experimente kann zwar im Allgemeinen keine gute Parametereinstellung für lange Experimente gefunden werden, es ist jedoch zu hoffen, dass besonders schlecht Einstellungen dadurch vermieden werden. Die Parameterkonfiguration, die sich bei den kurzen Läufen als am besten herausgestellt hat, ist in Tabelle 4.1 zusammengefasst. Wir werden sie im Folgenden mit K_L bezeichnen.

Parameter	Wert
Populationsgröße	40
Poolgröße	20
Crossover-Wahrscheinlichkeit	0,8
Mutations-Wahrscheinlichkeit	0,01

Tabelle 4.1: Parameterkonfiguration K_L des Langzeitexperiments.

Als Abbruchkriterium für die langen Läufe wurden 150000 Fitnessauswertungen gewählt. Aufgrund der in den kürzeren Experimenten gesammelten Erfahrungen war zu erwarten, dass bei diesen Einstellungen die längsten Läufe auf den zur Verfügung stehenden Rechner der Rechenanlage A2 nicht länger als 24 Stunden dauern sollten. Die zufällig aus der Rome-Library ausgewählten Graphen wurden jeweils vier Mal mit diesen Einstellungen optimiert. Die Ergebnisse der Läufe sind in Tabelle 4.2 aufgelistet. In der letzten Spalte ist zum Vergleich der Werte das Ergebnis der Variante PQ100-VAR-ALL-PERM20 aus der Vergleichsstudie von Mutzel und Gutwenger [15] als „Konkurrenz" aufgeführt.

Graph	$\|N\|$	$\|E\|$	Ergebnisse				Durchschnitt	„Konkurrenz"
grafo10113.94	94	139	68	65	62	65	65	65
grafo10306.95	95	129	20	21	21	22	21	22
grafo10426.100	100	138	26	28	28	26	27	31
grafo11286.92	92	17	9	9	10	10	9,5	10
grafo2271.71	71	89	7	7	7	7	7	7
grafo3532.99	99	154	84	93	82	82	85,25	89
grafo4444.52	52	73	6	6	6	6	6	8
grafo5745.50	50	76	15	15	14	14	14,5	16
grafo5996.78	78	103	12	12	12	12	12	12
grafo6523.47	47	65	7	7	7	7	7	8
grafo7918.84	84	135	54	53	51	53	52,75	58
grafo8029.75	75	105	17	17	18	19	17,75	17
grafo8065.82	82	118	28	26	27	27	27	30
grafo9169.67	67	106	36	35	35	33	34,75	35
grafo9239.62	62	85	6	6	6	6	6	7
grafo9600.65	65	82	4	4	4	4	4	4
grafo9841.70	70	97	16	16	15	15	15,5	17

Tabelle 4.2: Ergebnisse des Langzeitexperiments

Die folgenden Erkenntnisse kann man direkt aus den Daten ziehen:

- Die Graphen, bei denen nach der Berechnung 12 oder weniger Kreuzungen übrig bleiben, haben bei allen vier Läufen dasselbe Ergebnis (grafo2271.71, grafo444.52, grafo5996.78, grafo6523.47, grafo9239.62 und grafo9600.65).
- Bei den meisten Graphen ist das schlechteste Ergebnis der vier Läufe nicht viel schlechter als das beste Ergebnis.
- Beim Graph grafo 10113.94 ist das beste Ergebnis mit 62 Kreuzungen relativ weit von den anderen Läufen entfernt. Insbesondere war der erste Lauf der Schlechteste mit 68 Kreuzungen. Wären nicht mehrere Läufe durchgeführt worden, wäre dies also das Endergebnis.
- Beim Graph grafo3532.99 gibt es einen besonders schlechten Lauf als „Ausreißer". Das beste Ergebnis wurde jedoch gleich in 2 Läufen erhalten.
- Der arithmetische Mittelwert der Ergebnisse ist immer gleich gut oder besser als das Ergebnis der Variante PQ100-VAR-ALL-PERM20. [15].

Der GA kommt also offensichtlich auch ohne systematisch Optimierung der Laufzeitparameter schon zu sehr guten Ergebnissen. Allerdings darf man dabei nicht vergessen, dass die Rechenzeit für die 150000 Fitnessauswertungen unverhältnismäßig hoch ist im Vergleich zu der Rechenzeit der Vergleichsvariante. Während der GA bei der Langzeituntersuchung bis zu 12 Stunden rechnete, war die Vergleichsvariante bei allen Beispielen in weniger als einer Minute fertig, meist sogar in wenigen Sekunden. Es stellt sich also die Frage, ob es überhaupt notwendig ist, den GA so lange rechnen zu lassen. Es ist ja durchaus möglich, dass die endgültigen Ergebnisse bereits viel früher zur Verfügung stehen. Um dies zu untersuchen, wurde bei einigen Experimenten der Fitnesswert des besten Individuums in regelmäßigen Intervallen ausgegeben. Der Verlauf der Fitnesswerte ausgewählter Graphen bei der Optimierung ist in Abbildung 4.3 dargestellt.

Es ist klar zu erkennen, dass der Endwert der Optimierung meist schon in der ersten Hälfte des Laufs gefunden wurde. Weiterhin kann abgeschätzt werden, dass die Ergebnisse der Vergleichsvariante PQ100-VAR-ALL-PERM20 oft schon mit den ersten 10000 Fitnessauswertungen erreicht werden können.

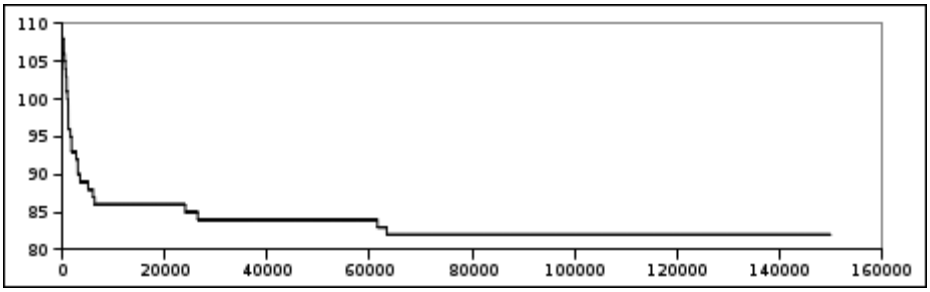
Das Langzeitexperiment hat damit seinen Zweck erfüllt, die Frage nach der Anzahl der notwendigen Fitnessauswertungen zu beantworten. Nach 10000 Fitnessauswertungen können wir schon ohne systematische Parameteroptimierung in vielen Fällen gleich gute oder bessere Ergebnisse erwarten, wie die zum Vergleich herangezogenen Variante PQ100-VAR-ALL-PERM20. Weiterhin kann aufgrund der Erfahrungen, die im Langzeitexperiment gesammelt wurden, davon ausgegangen werden, dass auf durschnittlichen aktuellen Rechensystemen die Optimierungszeit pro Lauf eine Stunde meist nicht überschritten wird.

4.3 Sequential Parameter Optimization

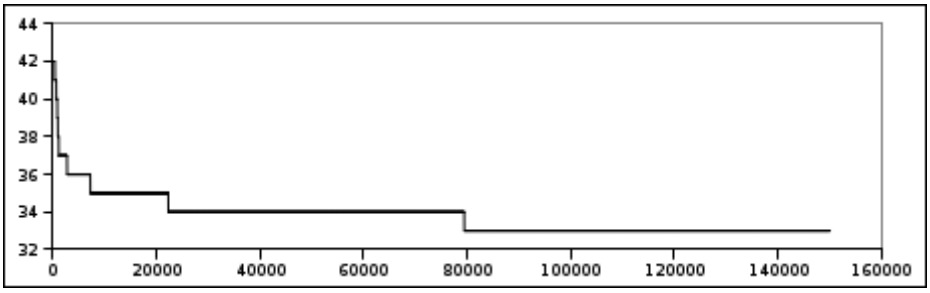
In diesem Abschnitt wird beschrieben, wie die SPO-Methode, die in Kapitel 2.5 beschrieben ist, auf die Variante VAR des GAs angewandt wurde, um die Laufzeitparameter zu optimieren. Das Ziel dieser Optimierung war, eine Konfiguration für den GA zu finden, mit der innerhalb von 10000 Fitnessauswertungen ein möglichst gutes Ergebnis gefunden wird. Die Darstellung der Parameteroptimierung erfolgt analog zu Kapitel 2.5.

(S-1) *Vorexperimentelle Planung*
Ein Teil der vorexperimentellen Untersuchung ist bereits im letzten Abschnitt beschrieben: Es wurde in sehr langen Testläufen festgestellt, dass 10000 Fitnessauswertungen zu guten Ergebnissen führen können. Dazu wurde eine mehr oder weniger zufällige Einstellung der Parameter vorgenommen. Um systematisch zu einer guten Konfiguration zu kommen, mussten zunächst die Faktoren identifiziert werden, die einen Einfluss auf das Experiment haben. Da es sich bei dem untersuchten GA um eine recht schlichte Variante handelt, waren lediglich vier Parameter zu betrachten. Tabelle 4.3 gibt eine Übersicht.

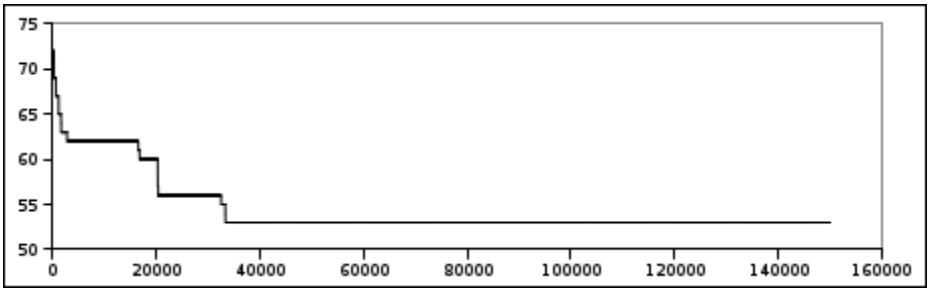
(S-2) *Wissenschaftlichen Behauptung*
Das Ziel des Experiments war, eine bessere Konfiguration der Laufzeitparameter zu finden. Die dahinter steckende Behauptung lautet, dass es eine Konfiguration



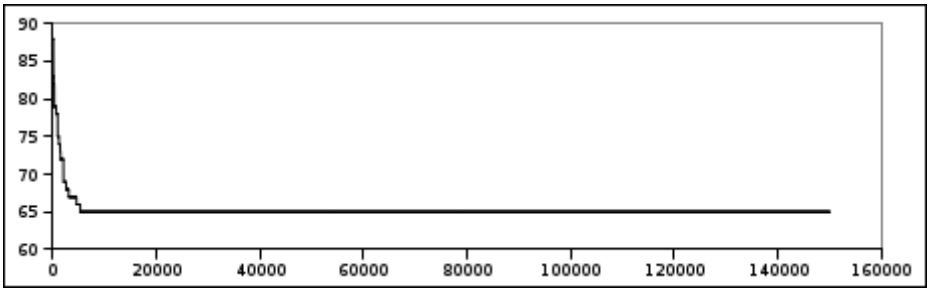
(a) Fitnessverlauf bei grafo3532.99



(b) Fitnessverlauf bei grafo9169.67



(c) Fitnessverlauf bei grafo7918.84



(d) Fitnessverlauf bei grafo10113.94

Abbildung 4.3: Verlauf der Fitnesswerte ausgewählter Graphen bei der Optimierung.

Parameter	Typ	Wertebereich
Populationsgröße	Integer	25–100
Poolgröße	Integer	12–50
Crossover-Wahrscheinlichkeit	Real	0,7–0,9
Mutations-Wahrscheinlichkeit	Real	0,005–0,5

Tabelle 4.3: Die relevante Faktoren für die SPO mit ihren Wertebereichen.

gibt, die besser ist, als die zuvor verwendeten. Gemäß der vorexperimentellen Untersuchung kann die Behauptung konkreter formuliert werden als:

> „Es gibt eine Parameterkonfiguration K^* für die Variante VAR, die bei 10000 Fitnessauswertungen im Mittel zu besseren Ergebnissen für einen ausgewählten Graphen kommt, als die bisher verwendete Konfiguration K_L."

(S-3) *Statistische Hypothese*
Die statistische Hypothese für das Experiment ist die Folgende:

> „Die Variante VAR kommt bei Parameterkonfiguration K^* im Mittel auf deutlich bessere Ergebnisse als mit der Paramterkonfiguration K_L."

(S-4) *Spezifikation* Ein großer Teil der Spezifikation ist bereits in den vorherigen Abschnitten beschrieben. Die folgende Aufzählung fasst die entscheidenden Punkte noch einmal zusammen:

1. Das Optimierungsproblem, namentlich das Kreuzungsminimierungsproblem, wurde in Abschnitt 2.4 beschrieben. Es soll die Anzahl der Dummy-Knoten und somit die Anzahl der daraus resultierenden Kreuzungen beim Erstellen einer Zeichnung eines Graphen minimiert werden.
2. Weitere Nebenbedingungen sind nicht zu beachten.
3. Die Initialisierungsmethode des GAs wurde in Abschnitt 3.2 beschrieben. Die Individuen werden zufällig initialisiert.
4. Als Abbruchmethode wurde die Anzahl der Fitnessauswertungen gewählt. In Abschnitt 4.2 wurde ermittelt, dass nach 10000 Fitnessauswertungen mit guten Ergebnissen gerechnet werden kann.
5. Der verwendete Algorithmus ist die in Abschnitt 3.2 vorgestellte Variante VAR des entwickelten GAs. Die Faktoren mit den zu untersuchenden Wertebereichen sind in Tabelle 4.3 dargestellt.
6. Das initiale experimentelle Design wurde mit Hilfe eines Skriptes aus dem beschriebenen Softwarepaket von Bartz-Beielstein erstellt [4]. Es wurden 40 verschiedene Konfigurationen erzeugt. Mit jeder Konfiguration werden jeweils vier Läufe durchgeführt.
7. Als Bewertungsmaß für die Performance wird letztlich die durchschnittlich erreichte Kreuzungszahl der durchgeführten Läufe dienen.

(S-5) *Experimentieren*
Die Versuche wurden auf der Rechenanlage A2 durchgeführt. Als Testobjekt

kam der Graph *grafo3532.99* zum Einsatz. Dieser schien aus mehreren Gründen ein gute Wahl zu sein:

- Im Langzeitexperiment traten die größten Unterschiede in den Ergebnissen auf.
- Abbildung 4.3(a) zeigt, dass es viele unterschiedliche Stufen gibt, auf denen das beste Individuum der Population über mehrere Generationen konstant bleibt. Insbesondere sind einige dieser Intervalle recht lang, so dass davon ausgegangen werden kann, dass dieser Graph schwierig zu optimieren ist. Dies ist für den Zweck des Experiments ideal. Es kann davon ausgegangen werden, dass ein leicht zu optimierender Graph bei vielen Versuchen zum gleichen Ergebnis führt, nämlich zum Optimum. Dann kann aber nicht mehr entschieden werden, welche der in den Versuchen verwendeten Parameterkonfigurationen die Beste war.

Nachdem die Versuche durchgeführt wurden, mussten die Ergebnisse zusammengetragen und in das passende Format für die Weiterverarbeitung im nächsten Schritt gebracht werden.

(S-6) *Statistische Modellierung und Vorhersagen*
Die statistische Modellierung und die Vorhersage wurden komplett durch das Softwarepaket vorgenommen. Es war lediglich notwendig, die gesammelten Daten in passender Form bereitzustellen.

(S-7) *Auswertung und Visualisierung*
Die Auswertung des Modells erfolgt ebenfalls durch die eingesetzte Software. Dabei wurde eine Vielzahl von Diagrammen erstellt, die angeben, wie groß der Fehler der Vorhersage ist und welche Effekte zwischen den Faktoren wirksam sind. Weiterhin wird die beste Parameterkonfiguration ausgegeben.

(S-8) *Optimierung*
Das Ergebnis der Auswertung durch das Softwarepaket von Bartz-Beielstein ist eine Menge von neuen Parameterkonfigurationen, bei denen aufgrund der Analyse am ehesten mit Verbesserungen zu rechnen ist.

(S-9) *Abbruch?*
Die Iteration über die Schritte (S-5) bis (S-10) wurde abgebrochen, nachdem drei Durchgänge hintereinander dieselbe Konfiguration als Beste aus der Bewertung bei der Analyse hervorging. In diesem Fall wurde mit Schritt (S-11) weitergemacht.

(S-10) *Anpassen des Designs*
Die Anpassung des Designs beschränkte sich darauf, die Ausgabe des Softwarepakets in die passende Form zu bringen, um sie als Parameterkonfiguration für neue Versuche verwenden zu können. Dabei ist zu beachten, dass die Anzahl der Wiederholungen des Versuchs so gewählt wird, dass bei der nächsten Auswertung alle Läufe gleich oft gewertet werden. Dies ist speziell dann nicht der Fall, wenn mehrfach hintereinander dieselbe Konfiguration als Beste aus der Analyse hervorgeht. So trat bei dem Experiment der Fall auf, dass bereits bei der ersten Iteration des SPO-Verfahrens die beste Konfiguration gefunden wurde. Bei den weiteren Iterationen musste daher die Wiederholungszahl der Versuche verdoppelt werden, da auch die Ergebnisse vorhergegangener Iterationen wieder mit

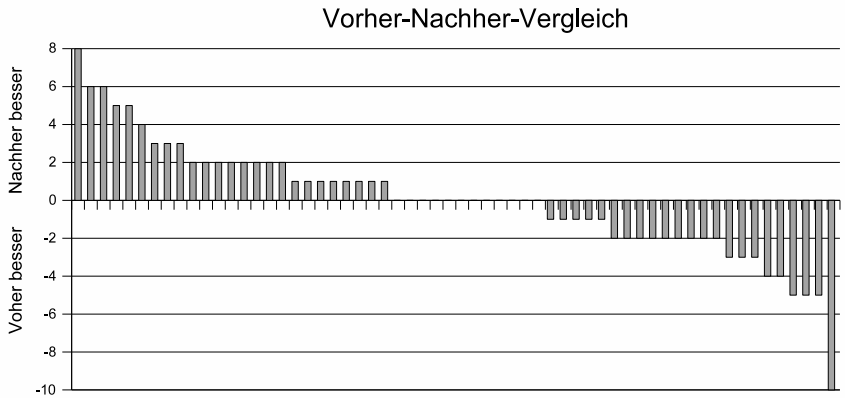


Abbildung 4.4: Differenzen der Kreuzungsanzahlen

in die Auswertung gelangen. Nach der Anpassung des Designs ging es mit der Durchführung der neuen Versuche in Schritt (S-5) weiter.

(S-11) *Verwerfen oder Akzeptieren der statistischen Hypothese*
Um die statistische Hypothese aus Schritt (S-2) zu testen, wurden 60 Graphen der Rome-Library zufällig ausgewählt und jeweils einmal mit der Parameterkonfiguration K_L aus dem Langzeitexperiment und einmal mit der optimierten Parameterkonfiguration K^* optimiert. Das Ergebnis dieses Versuchs ist in 4.4 dargestellt. Dazu wurden die Differenzen der Kreuzungsanzahlen gebildet und der Übersicht halber sortiert. Die Verteilung der Ergebnisse ist nahezu symmetrisch. Von den 60 Graphen hatte 25 mit der optimierten Parameterkonfiguration eine kleinere und 23 eine höhere Anzahl von Kreuzungen. Bei den 12 restlichen Graphen war die Anzahl der Kreuzungen gleich. Der arithmetische Mittelwert der Kreuzungen der berechneten Graphen ist mit den optimierten Parametern $32,7$ und mit den nicht optimierten Parametern $32,73$. Tabelle 4.4 fasst die Ergebnisse zusammen.

	K^*	Mittelwert der K_L
Anzahl besserer Ergebnisse	25	23
Mittelwert der Kreuzungen	32,7	32,73

Tabelle 4.4: Zusammenfassung der Ergebnisse

Damit ist die erhoffte signifikante Verbesserung durch die Optimierung der Parameter nicht eingetreten. Die statistische Hypothese, dass die Parameterkonfiguration K^* im Mittel zu deutlich besseren Ergebnissen kommt, muss verworfen werden.

(S-12) *Objektive Interpretation der Ergebnisse*

Dem Ziel, eine bessere Konfiguration der Laufzeitparameter als K_L zu finden, konnte das Experiment nicht gerecht werden. Es wurde vielmehr eine etwa gleich gute Konfiguration gefunden.

Diese Erkenntnis kann als Ausgangspunkt für weitere Untersuchungen dienen. Einerseits wurde die Optimierung der Parameter bisher nur anhand eines Graphens durchgeführt. Ein Experiment, das eine repräsentativere Teilmenge der Rome-Library in Betracht zieht, wäre ein denkbarer nächster Schritt. Weiterhin kann getestet werden, wie die hier verglichenen Parameterkonfigurationen im Vergleich zu anderen Konfigurationen abschneiden.

Da weitere Experimente zur Parameterkonfiguration im Rahmen dieser Arbeit aus zeitlichen Gründen nicht mehr vorgenommen werden können, wird im Folgenden die Konfiguration K^* verwendet werden.

Es kann für den Moment als legitim erachtet werden anzunehmen, dass bei Einsatz beider Konfigurationen brauchbare Ergebnisse erzielt werden können. Immerhin ist K_L aus einer Menge von Parameterkonfigurationen durch unsystematisches Testen als Beste hervorgegangen.

4.4 Vergleich der Varianten

Die bisherigen Experimente betrachteten immer nur die Variante VAR. In diesem Abschnitt wird nun ein Vergleich zwischen den vier Varianten durchgeführt, um festzustellen, welche die Beste ist. Die Untersuchung betrachtet dazu einerseits die Güte der ermittelten Lösungen und andererseits die Zeit, die benötigt wird, um die Berechnungen durchzuführen. Gegenstand der Betrachtung ist auch die Spannweite der Ergebnisse bei der wiederholten Durchführung der Planarisierung.

4.4.1 Planung des Experiments

Um die Varianten gut vergleichen zu können, wurde der Graph *grafo9092.61* aus der Rome-Library ausgewählt. Von diesem war bekannt, dass er schwierig genug zu planarisieren ist, dass nicht jedes Mal dasselbe Optimum gefunden wird. Außerdem hat sich in vorangegangenen Versuchen gezeigt, dass die Optimierung mit VAR auf diesem Graphen verglichen mit ähnlichen Graphen schneller ging. Dies war wichtig, damit die Berechnung aller vier Varianten auf demselben Rechner vorgenommen werden konnte, was die Vergleichbarkeit der Ausführungszeiten erst ermöglicht.

4.4.2 Durchführung des Experiments

Die vier Varianten FIX, VAR, WFIX und WVAR haben den Graphen *grafo9092.61* jeweils 30 Mal planarisiert. Das Programm hat bei der Berechnung neben dem Endergebnis, also der Anzahl der erreichten Kreuzungen, auch die verwendete CPU-Zeit für die spätere Auswertung ausgegeben. Alle vier Varianten haben dieselbe Parameterkonfiguraion K^* verwendet, die in Abschnitt 4.3 ermittelt wurde. Die Varianten, die Kantengewichte einsetzen, haben beide den Wert $\epsilon = 0,1$ verwendet, um die Kantengewichte zu erzeugen. Diese lagen also im Intervall $[0,9;1[$.

Das Rechensystem A1 hat alle Optimierungen durchgeführt. Dies hat den Vorteil, dass es nicht zu Unterschieden in der Rechenzeit kommt, die durch Unterschieden

zwischen den jeweiligen Rechner zustanden kommen. Zwar gibt es auch im Rechensystem A2 mehrere Rechner mit identischem Aufbau. Die Zuweisung der Aufgaben auf die Rechner geschieht jedoch automatisch und kann vom Benutzer nicht dahingehend beeinflusst werden, dass genau diese Rechner auch verwendet werden.

4.4.3 Diskussion der Ergebnisse

Abbildung 4.5 zeigt einen Boxplot der erreichten Ergebnisse. Die vier Varianten sind von oben nach unten in der Reihenfolge VAR, FIX, WVAR und WFIX dargestellt. Es ist deutlich zu erkennen, dass die Variante VAR die besten Ergebnisse liefert. Sie hat sowohl die niedrigste absolute Anzahl von Kreuzungen erreicht, als auch den niedrigsten Median aller Varianten. Auch die Spannweite der Ergebnisse hält sich in Grenzen, alle Ergebnisse liegen im Intervall $[34; 40]$. Das Histogramm in Abbildung 4.6(a) stellt die Verteilung der Ergebnisse der Variante VAR dar. Die Ergebnisse häufen sich am linken Rand des Intervalls, was darauf hindeutet, dass dort entweder das globale Optimum ist und es deshalb keine Lösung mit kleinerem Ergebnis gibt, oder dass Lösungen mit kleineren Ergebnissen so schwer zu finden sind, dass dies praktisch nicht vorkommt.

Am zweitbesten schneidet die Variante FIX ab. Die Spannweite der Werte ist zwar nicht viel weiter als bei der Variante VAR. Die Ergebnisse liegen im Intervall $[35, 42]$. Jedoch liegt der Median nicht wie bei VAR am unteren Rand des Intervalls, sondern recht zentral. Dies ist in dem dazugehörigen Histogramm aus Abbildung 4.6(b) ebenfalls deutlich zu erkennen.

Ganz anders sieht es bei den kantengewichteten Varianten aus. Die beste Lösung von WVAR ist gerade einmal so gut wie das schlechteste Ergebnis von FIX. Außerdem liegen die Ergebnisse im Intervall $[42; 78]$, die Spannweite ist also deutlich größer als bei den beiden Varianten ohne Kantengewichte. Weit abgeschlagen ist auch die Variante WFIX. Zwar ist die Streuung der Ergebnisse nicht ganz so weit wie bei WVAR, dafür werden aber auch nur vergleichsweise schlechte Lösungen im Intervall $[48, 73]$ gefunden. Das bedeutet, dass die beste von WFIX gefundenen Planarisierung 14 Kreuzungen mehr hat als die beste von VAR gefundene Lösung.

Das schlechte Abschneiden der kantengewichteten Varianten gibt einige Rätsel auf, die es noch zu lösen gilt. Für den Moment sollen die guten Ergebnisse der beiden anderen Varianten jedoch motivieren, mit weiteren Experimenten herauszufinden, wie gut deren Lösungen im Vergleich zu den bekannten, nicht evolutionären Methoden sind.

Tabelle 4.5 fasst die Ergebnisse noch einmal zusammen.

Die Laufzeiten der GAs unterscheiden sich start. Am schnellsten ist die Variante FIX, die im Mittel weniger als 5 Minuten benötig. Am langsamsten ist die Variante WVAR, die fast die 5-fache Zeit in Anspruch nimmt. Die folgende Auflistung zeigt die durchschnittlichen Laufzeiten in der Übersicht:

1. Variante FIX: $273,9$ Sekunden $= 4,57$ Minuten
2. Variante WFIX: $1012,34$ Sekunden $= 16,87$ Minuten
3. Variante VAR: $1019,44$ Sekunden $= 16,99$ Minuten

	VAR	FIX	WVAR	WFIX
Minimum	34	35	42	48
1. Quartile	34,25	37	49	57,5
Median	35,5	38	53,5	61,5
3. Quartile	37	38,75	57	67,75
Maximum	40	42	78	73
Arithmetisches Mittel	35,83	37,8	53,67	61,4

Tabelle 4.5: Die Ergebnisse des Vergleichsexperiments.

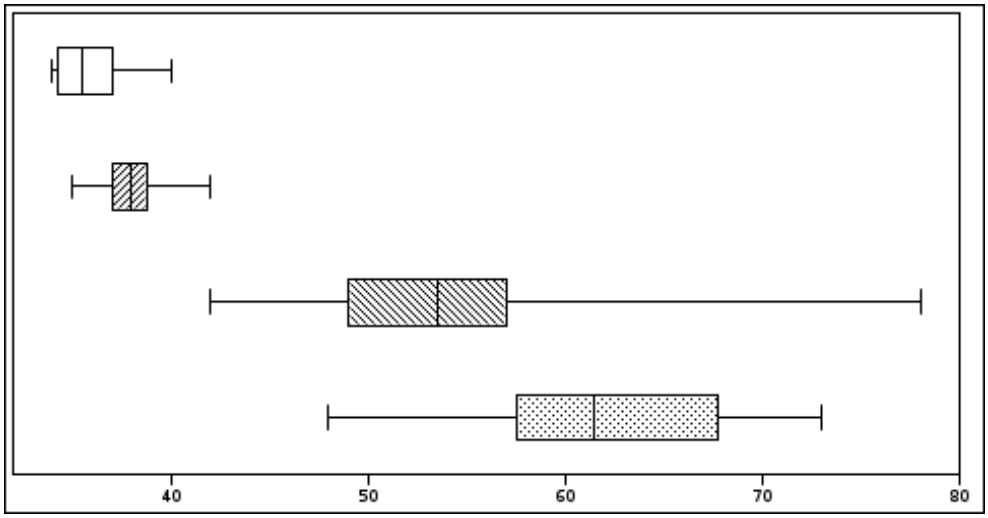


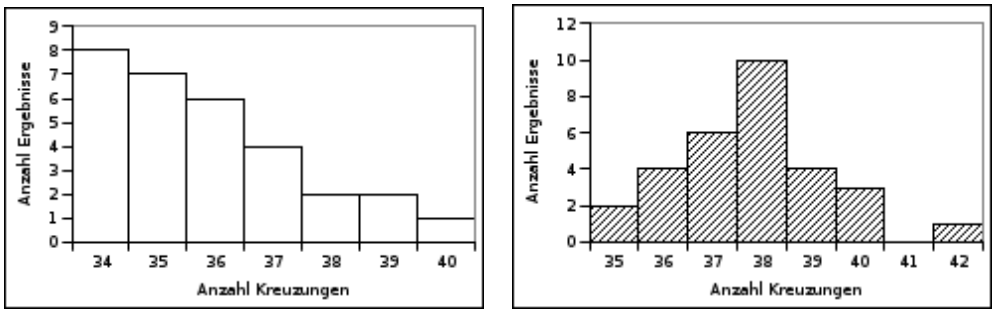
Abbildung 4.5: Übersicht der Ergebnisse der vier Varianten. Von oben nach unten: VAR, FIX, WVAR, WFIX

4. Variante WVAR: $1414,46$ Sekunden $= 23,57$ Minuten

Allgemein kann festgehalten werden, dass die Varianten, die mit fixer Einbettung arbeiten, schneller sind als solche mit variabler Einbettung. Weiterhin benötigen die Varianten, die mit Kantengewichten arbeiten, länger. Dies entspricht den auch den Erwartungen, denn das Kanteneinfügen bei variabler Einbettung ist aufwändiger als bei fixer Einbettung und die Suche nach kürzesten Wegen ist mit Kantengewichten aufwändiger als ohne.

Um zu illustrieren, in welcher Größenordnung sich diese Zeiten bewegen sei angemerkt, dass mit dem Verfahren PQ100-VAR-ALL20 in lediglich 5,125 Sekunden ein Ergebnis mit 34 Kreuzungen erreicht wird.

Soll eine Empfehlung für die Anwendung der hier getesteten GAs ausgesprochen werden, kann diese nur zu Gunsten der Variante VAR ausgehen. Zum einem ist sie die Einzige, die das beste bekannte Ergebnis von 34 Kreuzungen erreicht. Sie ist zwar langsamer als die Variante FIX, die in ihren Ergebnisse nicht allzu weit davon entfernt liegt, allerdings kann die Optimierungszeit beim Einsatz von GAs ohnehin nur eine untergeordnete Rolle spielen.



(a) Ergebnisse der Variante VAR

(b) Ergebnisse der Variante FIX

Abbildung 4.6: Verteilung der Ergebnisse bei den Varianten VAR und FIX.

4.5 Anwendung auf die Rome-Library

Die vorhergehenden Abschnitte beschreiben, wie sehr lange Läufe durchgeführt wurden, wie eine gute Parametereinstellung gesucht wurde und wie die vier verschiedenen Varianten in ihrer Leistungsfähigkeit zueinander stehen. Alle diese Experimente legend den Schwerpunkt auf die in dieser Arbeit vorgestellten GAs. Dieser Abschnitt soll nun die Relation zu schon bekannten Verfahren herstellen.

In der Vergleichsstudie von Gutwenger und Mutzel hat die Variante PQ100-VAR-ALL-PERM20 am besten abgeschnitten [15]. Die von ihr erreichten Ergebnisse werden aus diesem Grund zum Vergleich für die beste Variante des GAs, der Variante VAR, herangezogen.

4.5.1 Ermitteln der Testmenge

Als Testmenge dienten die Graphen der Rome-Library. Aufgrund der großen Menge der darin enthaltenen nicht planaren Graphen war es im Rahmen dieser Arbeit jedoch nicht möglich, diese alle zu Vergleichszwecken zu planarisieren. Die Testmenge wurde aus diesem Grund eingeschränkt auf die Graphen, die bei der Planarisierung mit dem Verfahren PQ100-VAR-ALL-PERM20 mehr als 14 Kreuzungen übrig behalten hatten. Dies ist auch legitim, da die meisten Graphen, die nicht in diese Gruppe fallen, mit einem anderen Verfahren wesentlich schneller planarisiert werden könnten. Dabei kann man davon ausgehen, das globale Optimum, die sogenannte *Kreuzungszahl*, oft in vielen Fällen zu erreichen. Anstatt dessen ein vergleichsweise langsames Verfahren wie den in dieser Arbeit vorgestellten GA einzusetzen, ist nicht im Sinne des ressourcenschonenden Umgangs mit der zur Verfügung stehenden Rechenzeit.

Wurden alle Graphen aussortiert, die weniger als 15 Kreuzungen bei der Planarisierung mit PQ100-VAR-ALL20 übrig behielten, bleibt eine Menge von 2339 Testgraphen übrig. Die Verteilung der Knotenanzahlen dieser Graphen ist in Abbildung 4.7 dargestellt. Jeder Balken in dem Diagramm repräsentiert die Anzahl der Graphen, welche die an der X-Achse markierte Knotenanzahl haben.

Die Verteilung der Kantenanzahlen unter den Testgraphen ist in Abbildung 4.8 dargestellt. Jeder Balken repräsentiert die Kantenanzahl, welche die Graphen mit der an

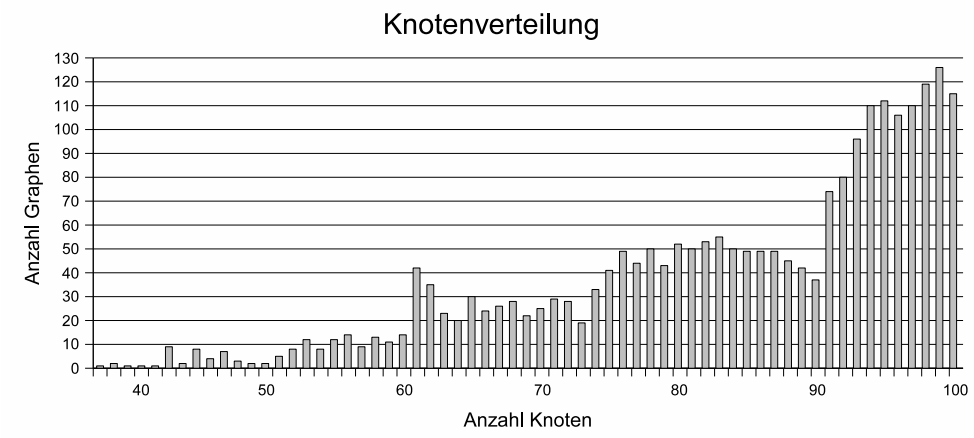


Abbildung 4.7: Verteilung der Knotenanzahlen unter den Graphen der Testmenge

der X-Achse markierten Knotenanzahl im Mittel haben.

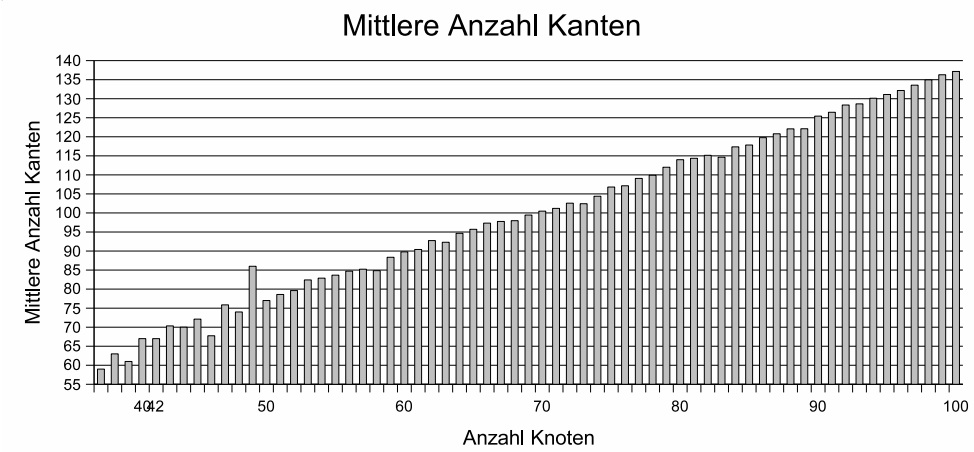


Abbildung 4.8: Verteilung der durchschnittlichen Kantenanzahl unter den Graphen der Testmenge

Es ist offensichtlich, dass sich die Verteilung der Kantenanzahlen im Vergleich zu der gesamten Rome-Library, die in Abbildung 4.2 dargestellt ist, nicht merklich verändert hat. Die Verteilung entspricht ist dem der gesamte Rome-Library ähnlich. Anders bei der Verteilung der Knotenanzahlen. Verglichen mit Abbildung 4.1 fehlen hier, wie gewünscht, die leichter zu planarisierenden Graphen. Dies führt zu einer merklichen Ausdünnung bei den kleineren Kontenanzahlen.

Für den Vergleich ist diese Ausdünnung jedoch ohne Bedeutung, solange darauf hingewiesen wird, dass die nicht in der Testmenge enthaltenen Graphen ohnehin nicht in den Hauptanwendungsbereich des GAs fallen.

4.5.2 Durchführung des Experiments

Aufgrund des hohen Rechenaufwands, den die Planarisierung durch den GA mit sich bringt, wurden die Berechnungen auf die beiden Systeme A1 und A2 verteilt. Jeder Lauf des GA wurde genau einmal gestartet. Es wurden also nicht wie bei dem Langzeitexperiment in Abschnitt 4.2 oder dem Vergleich der Varianten in Abschnitt 4.4 mehrere Läufe pro Graph durchgeführt. Dies hätte zwar voraussichtlich noch zu Verbesserungen der erreichten Resultate geführt, hätte aber einerseits den zeitlichen Rahmen gesprengt und andererseits den ohnehin schon großen Unterschied in den Berechnungszeiten zu der Vergleichsvariante erhöht.

4.5.3 Diskussion der Ergebnisse

Die Ergebnisse, die mit dem GA erzielt wurden, sind vielversprechend. Von den 2339 Graphen der Testmenge wurden 1419 mit weniger Kreuzungen planarisiert als mit dem Verfahren PQ100-VAR-ALL-PERM20, bei 430 Graphen war die Anzahl der Kreuzungen gleich und bei 430 Graphen war die Anzahl der Kreuzungen größer. In Abbildung 4.9 ist die mittlere Anzahl von Kreuzungen, die unter Graphen mit derselben Knotenzahl von den beiden Verfahren erreicht wurde, dargestellt. Es ist gut zu erkennen, dass die Werte der mittleren Anzahl von Kreuzungen des GAs bis auf eine Ausnahme immer unterhalb derer des Vergleichsverfahrens liegt.

Abbildung 4.10 stellt dar, wie viele Graphen jeweils eine bestimmte Anzahl von Kreuzungen erhalten haben. Besonders auffällig ist, dass lediglich der GA Ergebnisse unterhalb von 14 Kreuzungen erreicht hat.

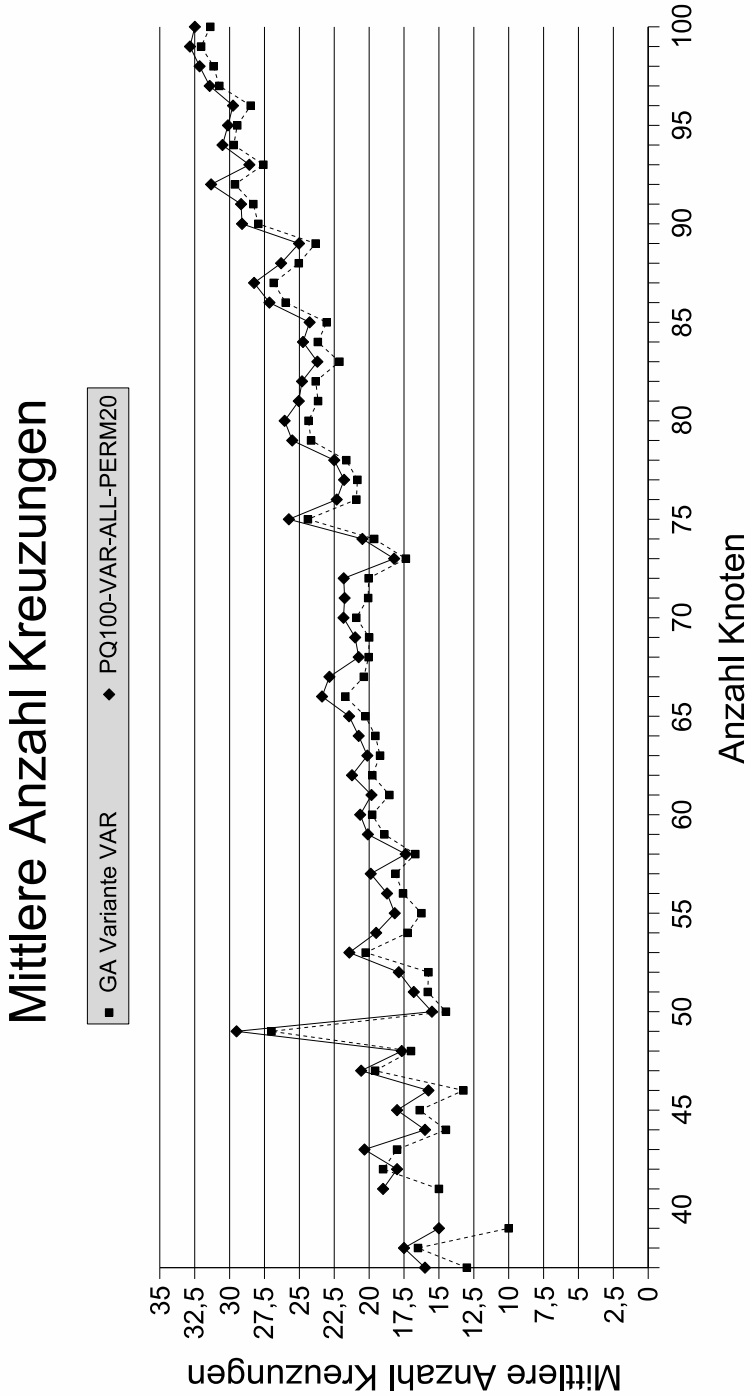


Abbildung 4.9: Darstellung der mittleren Anzahl von Kreuzungen aufgeteilt nach der Knotenanzahl der zugrunde liegenden Graphen.

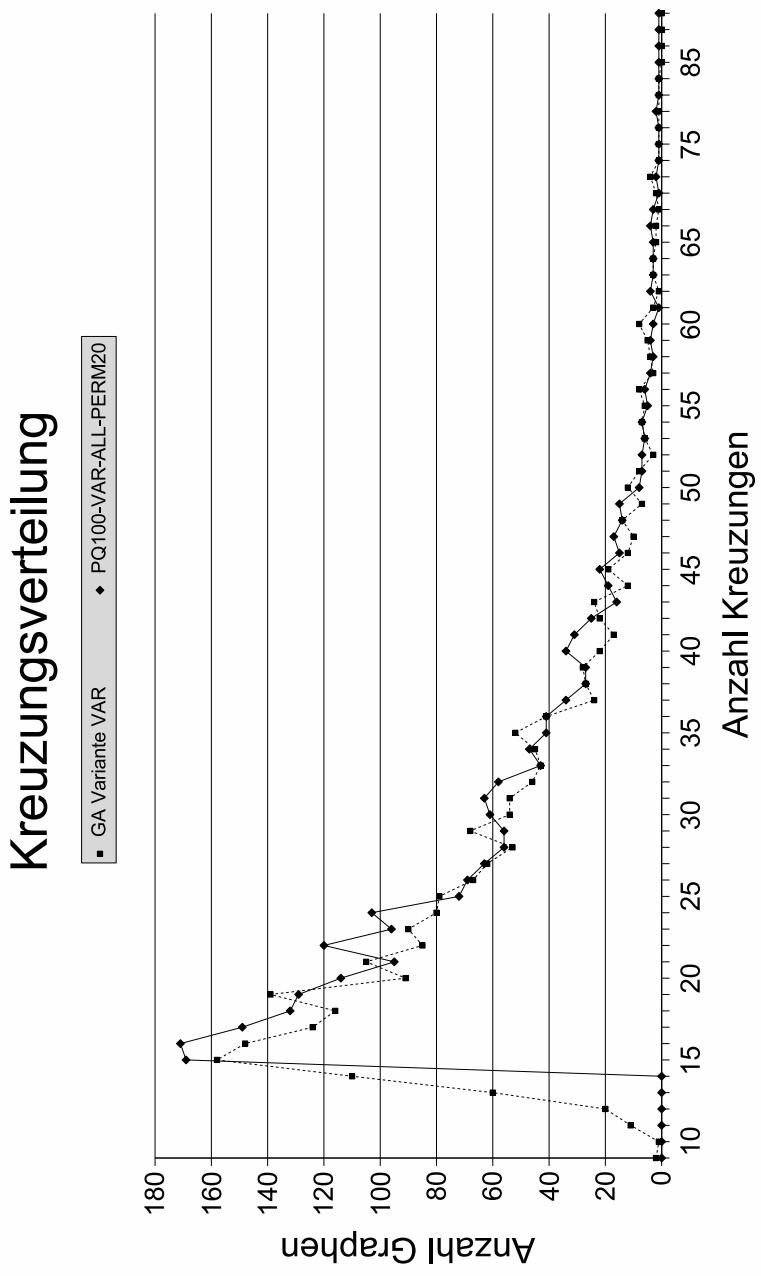


Abbildung 4.10: Anzahl der Graphen nach Anzahl der Kreuzungen.

Kapitel 5

Fazit / Ausblick

5.1 Fazit

Das Ziel dieser Arbeit war, einen Evolutionären Algorithmus zu entwerfen, der den Planarisierungsschritt des TSM-Modells durchführt. Mit dem Entwurf des vorgestellten Genetischen Algorithmus wurde dieses Ziel erreicht.

Zwar konnte durch systematisches Experimentieren keine Verbesserung der Lauzeitparameter-Einstellungen erreicht werden, dennoch führte eine der vier implementierten Varianten bei entsprechenden Voraussetzungen zu brauchbaren Ergebnissen.

Der GA in der Variante VAR kann daher für die Anwendung empfohlen werden, sofern genügend Zeit für die Optimierung bereitsteht. Sollte dies nicht der Fall sein, liefert das bisher führende Verfahren PQ100-VAR-ALL20 weiterhin sehr gute Ergebnisse. Für die praktische Anwendung lautet die Empfehlung, zunächst das Verfahren PQ100-VAR-ALL20 zu implementieren. Sollten dessen Ergebnisse für den Anwendungsfall nicht ausreichen und die Rechenzeit eine untergeordnete Rolle spielen, so kann die Variante VAR des GAs hinzugefügt werden. Dies ist auch von der Entwicklung her eine sinnvolle Vorgehensweise, da das Verfahren PQ100VAR-ALL20 ebenfalls als Herzstück das Kanteneinfügen bei variabler Einbettung verwendet, das in dem GA wiederverwendet werden kann.

5.2 Ausblick

Die Experimente, die mit dem Genetischen Algorithmus durchgeführt wurden, habe gezeigt, dass der in dieser Arbeit vorgestellte genetische Algorithmus ein vielversprechender Ansatz ist, der als Grundlage für weitere Untersuchungen dienen kann. Die folgende Aufzählung gibt eine unvollständige Übersicht über mögliche nächste Schritte:

- Die Berechnung interessanter Benchmark-Graphen beispielsweise aus der Rome-Library kann mit der Variante FIX wiederholt werden. Eine andere Möglichkeit

ist, die Anzahl der Fitnessauswertungen zu steigern. In den Verläufen der besten Fitnesswerte war zu erkennen, das auch jenseits der 10000 Fitnessauswertungen teilweise noch deutliche Verbesserungen der Ergebnisse zu verzeichnen waren. Die vorliegende Variante VAR könnte beispielsweise in einem länger angelegten Versuch einige der Experimente noch einmal mit 40000 Fitnessauswertungen durchführen.

- Die Fitnessauswertung kann durch den Einsatz von On-Line Verfahren zum Testen der Planarität effizienter gestaltet werden. In der beschriebenen Implementierung wird bei der Fitnessauswertung nach jedem Kanteneinfügen ein Planaritätstest in linearer Laufzeit durchgeführt. Dies ist jedoch nicht optimal, da das Verfahren jeweils nur eine Kante zu einem planaren Graphen hinzufügt. Dies ist eine günstige Situation, um ein On-Line Verfahren für den Planaritätstest einzusetzen. Dies würde es voraussichtlich ermöglichen, wesentlich mehr Fitnessauswertungen in derselben Zeit durchzuführen, was einerseits die Planarisierung von größeren Graphen in vertretbarer Zeit erlauben würde, andererseits bei Graphen gleicher Größe zu einer Verringerung der Laufzeit bzw. zu einer Verbesserung der Ergebnisse führen könnte.

- Bisher wurde die Ausgangspopulation rein zufällig initialisiert. Es kann untersucht werden, ob sich die erzielten Ergebnisse verbessern lassen, indem die Ausgangspopulation gezielt initialisiert wird.

- Die Auswirkungen von Multistarts, also der mehrfachen Ausführung des genetischen Algorithmus, kann weiter untersucht werden. Bei den Langzeituntersuchungen in Abschnitt 4.2 wurde deutlich, dass Multistarts ein vielversprechender Ansatz ist, um bessere Ergebnisse zu erzielen.

- Der Verlauf der besten Fitnesswerte, der in Abschnitt 4.2 dargestellt ist, lässt vermuten, dass lokal Optima ein Problem für den in dieser Arbeit vorgestellten genetischen Algorithmus sind. Es kann daher lohnenswert sein, verschiedene Diversifikationsmethoden zu erproben, um schneller aus der Falle der lokalen Optima zu entweichen.

- Die Auswirkung von adaptiven Laufzeitparametern kann untersucht werden.

- Die kantengewichteten Varianten des GAs hatten im Experiment unerwartet hohen Anzahlen von Kreuzungen. Es erscheint daher sinnvoll, diese noch einmal genauer zu betrachten und herauszufinden, ob Kantengewichte prinzipiell für die Aufgabe ungeeignet sind oder ob lediglich die spezielle Implementierung für das schlechte Abschneiden verantwortlich ist.

- Es stehen noch eine Reihe weiterer evolutionärer Operatoren für Genetische Algorithmen mit Permutationen zur Verfügung. Es kann untersucht werden, wie die Wahl der Operatoren sich auf das Ergebnis auswirkt.

Zusammenfassend lässt sich sagen, dass mit dem vorgestellten genetischen Algorithmus ein interessanter Weg eingeschlagen wurde, der für weitere Untersuchungen lohnenswert erscheint.

Literaturverzeichnis

[1] Altner, G. *Darwin, seine Theorie und ihr Zustandekommen.* Wissenschaftliche Buchgesellschaft Darmstadt, 1981, ch. 1, pp. 5–8.

[2] Ayala, F., and Valentine, J. *Evolving: The theory and process of organic evolution.* Benjamin, Menlo Park, CA, 1979.

[3] Bartz-Beielstein, T. *New Experimentalism Applied to Evolutionary Computation.* PhD thesis, Universität Dortmund, Dortmund, 2005.

[4] Bartz-Beielstein, T. *Experimental Research in Evolutionary Computation – The New Experimentalism.* Natural Computing Series. Springer, Berlin, 2006. (im Druck).

[5] Battista, G. D., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., and Vargiu, F. An experimental comparison of four graph drawing algorithms. *Computational Geometry Therory and Applications 7* (1997), 303–326.

[6] Bodendiek, R., and Lang, R. *Lehrbuch der Graphentheorie, Band 1.* Spektrum, 1995.

[7] Box, G. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics 6, 2* (1957), 81–101.

[8] Box, G. E. P., Hunter, W. G., and Hunter, J. S. *Statistics for Experimenters.* John Wiley & Sons, 1978.

[9] Branke, J., Bucher, F., and Schmeck, H. Using genetic algorithms for drawing undirected graphs. Tech. Rep. 347, D-76128 Karlsruhe, Germany, 1996.

[10] Diestel, R. *Graph Theory*, 2. ed. Springer, 2000.

[11] Friedberg, R. A learning machine - part I. *IBM Journal of Research and Development 2, 1* (1958), 2–13.

[12] Friedberg, R., Dunham, B., and North, J. A learning machine - part II. *IBM Journal of Research and Development 3, 3, (Juli)* (1959), 282–287.

[13] Goldberg, D. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, 1989.

[14] Gutwenger, C., and Mutzel, P. A linear time implementation of spqr-trees. In *GD 2000* (2001), J. Marks, Ed., LNCS 1984, Springer, pp. 77–90.

[15] Gutwenger, C., and Mutzel, P. An experimental study of crossing minimazation heuristics. In *GD 2003* (2004), G. Liotta, Ed., LNCS 2912, pp. 13–24.

[16] Gutwenger, C., Mutzel, P., and Weiskircher, R. Inserting an edge into a planar graph. *Algorithmica 41*, 41 (2005), 289–308.

[17] Holland, J. *Adaptation in natural and artificial Systems.* Univ. of Michigan, NN, 1975.

[18] Hopcroft, J., and Tarjan, R. Efficient planarity testing. *J. ACM 21*, 4 (1974), 549–568.

[19] I. Gerdes, F. Klawonn, R. K. *Evolutionäre Algorithmen.* Computational Intelligence. Vieweg, Wiesbaden, 2004.

[20] Johnson, M. R., and Johnson, D. S. Crossing number is np-complete. *SIAM J. Algebraic Discrete Methods 4*, 3 (1983), 312–316.

[21] King, R. C., and Stansfield, W. D. *A Dictionary of Genetics*, 6. ed. Oxford University Press, 2002.

[22] Koza, J. R. *Genetic Programming: On the Programming of Computers by the Means of Natural Selection.* The MIT Press, Cambridge, MA, 1992.

[23] Lawler, E. L., Ed. *The traveling salesman problem.* Wiley, 1987.

[24] Lophaven, S. N., Nielsen, H. B., and Søndergaard. Aspects of the matlab toolbox dace. Tech. Rep. IMM-REP-2002-13, Technical University of Denmark, 2002.

[25] Lophaven, S. N., Nielsen, H. B., and Søndergaard, J. Dace – a matlab kriging toolbox. Tech. Rep. IMM-REP-2002-12, Technical University of Denmark, 2002.

[26] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, 2 ed. Springer, Berlin, 1994.

[27] Montgomery, D. C. *Design and Analysis of Experiments.* John Wiley & Sons, 1997.

[28] Nissen, V. *Einführung in Evolutionäre Algorithmen Optimierung nach dem Vorbild der Evolution.* Vieweg, Göttingen, 1997.

[29] Patrignani, M. On the complexity of orthogonal compaction. *Comput. Geom. Theory Appl. 19*, 1 (2001), 47–67.

[30] Purchase, H. Which aesthetic has the greatest effect on human understanding? In *Graph Drawing (Proc. GD '97)* (1998), G. Di Battista, Ed., LNCS 1353, Springer-Verlag, pp. 248–261.

[31] Purchase, H. C., Cohen, R. F., and James, M. Validating graph drawing aesthetics. In *Graph Drawing (Proc. GD '95)* (1996), F. J. Brandenburg, Ed., LNCS 1027, Springer-Verlag, pp. 435–446.

[32] Rosete-Suarez, A., Sebag, M., and Ochoa-Rodriguez, A. A study of evolutionary graph drawing. Tech. Rep. LRI-RR-1228, Université Paris-Sud XI, 91405 Orsay-DEDEX, France, 1999.

[33] Santner, T., Williams, B., and Notz, W. *The Design and Analysis of Computer Experiments*. Springer, 2003.

[34] Spiegel, M. R., and Stephens, L. J. *Statistik*. mitp-Verlag, 2003.

[35] Wegener, I. *Theoretische Informatik*. Teubner, Stuttgart, 1993.

[36] Whitley, D. *Handbook of Evolutionary Computation*. IOP Publishing and Oxford University Press, New York and Bristol (UK), 1997, ch. Permutations, pp. C 3.3.3:1–15.